



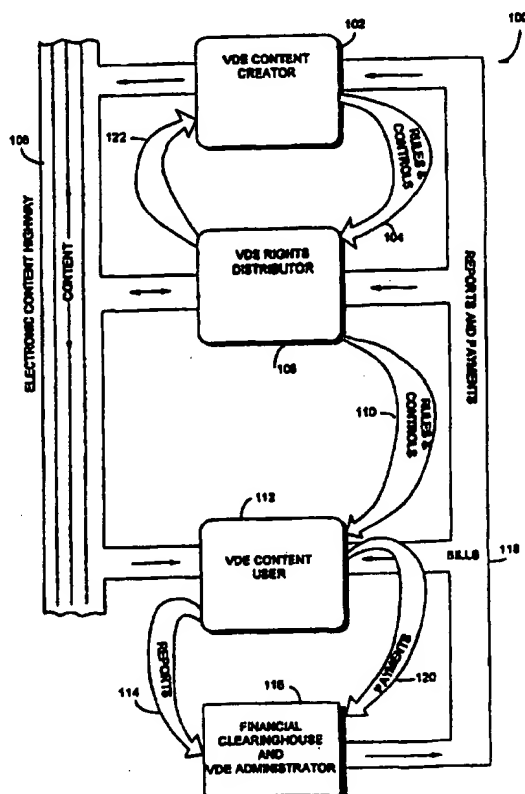
INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 1/00	A1	(11) International Publication Number: WO 98/09209 (43) International Publication Date: 5 March 1998 (05.03.98)
<p>(21) International Application Number: PCT/US97/15243</p> <p>(22) International Filing Date: 29 August 1997 (29.08.97)</p> <p>(30) Priority Data: 08/706,206 30 August 1996 (30.08.96) US</p> <p>(71) Applicant: INTERTRUST TECHNOLOGIES CORP. (US/US); 460 Oakmead Parkway, Sunnyvale, CA 94086 (US).</p> <p>(72) Inventors: GINTER, Karl, L.; 10404 43rd Avenue, Beltsville, MD 20705 (US). SHEAR, Victor, H.; 5203 Battery Lane, Bethesda, MD 20814 (US). SIBERT, W., Olin; 30 Ingleside Road, Lexington, MA 02173-2522 (US). SPAHN, Francis, J.; 2410 Edwards Avenue, El Cerrito, CA 94530 (US). VAN WIE, David, M.; 1250 Lakeside Drive, Sunnyvale, CA 94086 (US).</p> <p>(74) Agent: FARIS, Robert, W.; Nixon & Vanderhye P.C., 8th floor, 1100 North Glebe Road, Arlington, VA 22201-4714 (US).</p>	<p>(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG).</p> <p>Published <i>With international search report.</i> <i>Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.</i></p>	

(54) Title: **SYSTEMS AND METHODS FOR SECURE TRANSACTION MANAGEMENT AND ELECTRONIC RIGHTS PROTECTION**

(57) Abstract

The present invention provides systems and methods for electronic commerce including secure transaction management and electronic rights protection. Electronic appliances such as computers employed in accordance with the present invention help to ensure that information is accessed and used only in authorized ways, and maintain the integrity, availability, and/or confidentiality of the information. Secure subsystems used with such electronic appliances provide a distributed virtual distribution environment (VDE) that may enforce a secure chain of handling and control, for example, to control and/or meter or otherwise monitor use of electronically stored or disseminated information. Such a virtual distribution environment may be used to protect rights of various participants in electronic commerce and other electronic or electronic-facilitated transactions. Secure distributed and other operating system environments and architectures, employing, for example, secure semiconductor processing arrangements that may establish secure, protected environments at each node. These techniques may be used to support an end-to-end electronic information distribution capability that may be used, for example, utilizing the "electronic highway".



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LJ	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

SYSTEMS AND METHODS FOR SECURE TRANSACTION MANAGEMENT AND ELECTRONIC RIGHTS PROTECTION

Field(s) of the Invention(s)

This invention generally relates to computer and/or electronic security.

5

More particularly, this invention relates to systems and techniques for secure transaction management. This invention also relates to computer-based and other electronic appliance-based technologies that help to ensure that information is
10 accessed and/or otherwise used only in authorized ways, and maintains the integrity, availability, and/or confidentiality of such information and processes related to such use.

The invention also relates to systems and methods for
15 protecting rights of various participants in electronic commerce and other electronic or electronically-facilitated transactions.

The invention also relates to secure chains of handling and control for both information content and information employed to
20 regulate the use of such content and consequences of such use. It also relates to systems and techniques that manage, including meter and/or limit and/or otherwise monitor use of electronically stored and/or disseminated information. The invention

particularly relates to transactions, conduct and arrangements that make use of, including consequences of use of, such systems and/or techniques.

5 The invention also relates to distributed and other operating systems, environments and architectures. It also generally relates to secure architectures, including, for example, tamper-resistant hardware-based processors, that can be used to establish security at each node of a distributed system.

10

Background and Summary of the Invention(s)

Telecommunications, financial transactions, government processes, business operations, entertainment, and personal business productivity all now depend on electronic appliances.

15 Millions of these electronic appliances have been electronically connected together. These interconnected electronic appliances comprise what is increasingly called the "information highway." Many businesses, academicians, and government leaders are concerned about how to protect the rights of citizens and

20 organizations who use this information (also "electronic" or "digital") highway.

Electronic Content

Today, virtually anything that can be represented by words, numbers, graphics, or system of commands and instructions can be formatted into electronic digital information.

5 Television, cable, satellite transmissions, and on-line services transmitted over telephone lines, compete to distribute digital information and entertainment to homes and businesses. The owners and marketers of this content include software developers, motion picture and recording companies, publishers
10 of books, magazines, and newspapers, and information database providers. The popularization of on-line services has also enabled the individual personal computer user to participate as a content provider. It is estimated that the worldwide market for electronic information in 1992 was approximately \$40 billion and
15 is expected to grow to \$200 billion by 1997, according to Microsoft Corporation. The present invention can materially enhance the revenue of content providers, lower the distribution costs and the costs for content, better support advertising and usage information gathering, and better satisfy the needs of
20 electronic information users. These improvements can lead to a significant increase in the amount and variety of electronic information and the methods by which such information is distributed.

The inability of conventional products to be shaped to the needs of electronic information providers and users is sharply in contrast to the present invention. Despite the attention devoted by a cross-section of America's largest telecommunications, computer, entertainment and information provider companies to some of the problems addressed by the present invention, only the present invention provides commercially secure, effective solutions for configurable, general purpose electronic commerce transaction/distribution control systems.

10

Controlling Electronic Content

The present invention provides a new kind of "virtual distribution environment" (called "VDE" in this document) that secures, administers, and audits electronic information use. VDE also features fundamentally important capabilities for managing content that travels "across" the "information highway." These capabilities comprise a rights protection solution that serves all electronic community members. These members include content creators and distributors, financial service providers, end-users, and others. VDE is the first general purpose, configurable, transaction control/rights protection solution for users of computers, other electronic appliances, networks, and the information highway.

15

20

A fundamental problem for electronic content providers is extending their ability to control the use of proprietary information. Content providers often need to limit use to authorized activities and amounts. Participants in a business model involving, for example, provision of movies and advertising on optical discs may include actors, directors, script and other writers, musicians, studios, publishers, distributors, retailers, advertisers, credit card services, and content end-users. These participants need the ability to embody their range of agreements and requirements, including use limitations, into an "extended" agreement comprising an overall electronic business model. This extended agreement is represented by electronic content control information that can automatically enforce agreed upon rights and obligations. Under VDE, such an extended agreement may comprise an electronic contract involving all business model participants. Such an agreement may alternatively, or in addition, be made up of electronic agreements between subsets of the business model participants. Through the use of VDE, electronic commerce can function in the same way as traditional commerce—that is commercial relationships regarding products and services can be shaped through the negotiation of one or more agreements between a variety of parties.

Commercial content providers are concerned with ensuring proper compensation for the use of their electronic information. Electronic digital information, for example a CD recording, can today be copied relatively easily and inexpensively. Similarly, unauthorized copying and use of software programs deprives rightful owners of billions of dollars in annual revenue according to the International Intellectual Property Alliance. Content providers and distributors have devised a number of limited function rights protection mechanisms to protect their rights. Authorization passwords and protocols, license servers, "lock/unlock" distribution methods, and non-electronic contractual limitations imposed on users of shrink-wrapped software are a few of the more prevalent content protection schemes. In a commercial context, these efforts are inefficient and limited solutions.

Providers of "electronic currency" have also created protections for their type of content. These systems are not sufficiently adaptable, efficient, nor flexible enough to support the generalized use of electronic currency. Furthermore, they do not provide sophisticated auditing and control configuration capabilities. This means that current electronic currency tools lack the sophistication needed for many real-world financial business models. VDE provides means for anonymous currency

and for "conditionally" anonymous currency, wherein currency related activities remain anonymous except under special circumstances.

5

VDE Control Capabilities

VDE allows the owners and distributors of electronic digital information to reliably bill for, and securely control, audit, and budget the use of, electronic information. It can reliably
10 detect and monitor the use of commercial information products. VDE uses a wide variety of different electronic information delivery means: including, for example, digital networks, digital broadcast, and physical storage media such as optical and magnetic disks. VDE can be used by major network providers,
15 hardware manufacturers, owners of electronic information, providers of such information, and clearinghouses that gather usage information regarding, and bill for the use of, electronic information.

20

VDE provides comprehensive and configurable transaction management, metering and monitoring technology. It can change how electronic information products are protected, marketed, packaged, and distributed. When used, VDE should result in higher revenues for information providers and greater

user satisfaction and value. Use of VDE will normally result in lower usage costs, decreased transaction costs, more efficient access to electronic information, re-usability of rights protection and other transaction management implementations, greatly
5 improved flexibility in the use of secured information, and greater standardization of tools and processes for electronic transaction management. VDE can be used to create an adaptable environment that fulfills the needs of electronic information owners, distributors, and users; financial
10 clearinghouses; and usage information analyzers and resellers.

Rights and Control Information

In general, the present invention can be used to protect the rights of parties who have:

15

(a) proprietary or confidentiality interests in electronic information. It can, for example, help ensure that information is used only in authorized ways;

20

(b) financial interests resulting from the use of electronically distributed information. It can help ensure that content providers will be paid for use of distributed information; and

- (c) interests in electronic credit and electronic currency storage, communication, and/or use including electronic cash, banking, and purchasing.

5 Protecting the rights of electronic community members involves a broad range of technologies. VDE combines these technologies in a way that creates a "distributed" electronic rights protection "environment." This environment secures and protects transactions and other processes important for rights protection. VDE, for example, provides the ability to prevent, or
10 impede, interference with and/or observation of, important rights related transactions and processes. VDE, in its preferred embodiment, uses special purpose tamper resistant Secure Processing Units (SPUs) to help provide a high level of security
15 for VDE processes and information storage and communication.

The rights protection problems solved by the present invention are electronic versions of basic societal issues. These issues include protecting property rights, protecting privacy
20 rights, properly compensating people and organizations for their work and risk, protecting money and credit, and generally protecting the security of information. VDE employs a system that uses a common set of processes to manage rights issues in an efficient, trusted, and cost-effective way.

VDE can be used to protect the rights of parties who create electronic content such as, for example: records, games, movies, newspapers, electronic books and reference materials, personal electronic mail, and confidential records and communications.

5 The invention can also be used to protect the rights of parties who provide electronic products, such as publishers and distributors; the rights of parties who provide electronic credit and currency to pay for use of products, for example, credit clearinghouses and banks; the rights to privacy of parties who
10 use electronic content (such as consumers, business people, governments); and the privacy rights of parties *described* by electronic information, such as privacy rights related to information contained in a medical record, tax record, or personnel record.

15

In general, the present invention can protect the rights of parties who have:

- 20 (a) commercial interests in electronically distributed information -- the present invention can help ensure, for example, that parties, will be paid for use of distributed information in a manner consistent with their agreement;

- (b) proprietary and/or confidentiality interests in electronic information -- the present invention can, for example, help ensure that data is used only in authorized ways;

5

- (c) interests in electronic credit and electronic currency storage, communication, and/or use -- this can include electronic cash, banking, and purchasing; and

10

- (d) interests in electronic information derived, at least in part, from use of other electronic information.

VDE Functional Properties

15

VDE is a cost-effective and efficient rights protection solution that provides a unified, consistent system for securing and managing transaction processing. VDE can:

- (a) audit and analyze the use of content,

20

- (b) ensure that content is used only in authorized ways, and

- (c) allow information regarding content usage to be used only in ways approved by content users.

In addition, VDE:

5

- (a) is very configurable, modifiable, and re-usable;
- (b) supports a wide range of useful capabilities that may be combined in different ways to accommodate most potential applications;

10

- (c) operates on a wide variety of electronic appliances ranging from hand-held inexpensive devices to large mainframe computers;

15

- (d) is able to ensure the various rights of a number of different parties, and a number of different rights protection schemes, simultaneously;

20

- (e) is able to preserve the rights of parties through a series of transactions that may occur at different times and different locations;

- (f) is able to flexibly accommodate different ways of securely delivering information and reporting usage; and
- 5 (g) provides for electronic analogues to "real" money and credit, including anonymous electronic cash, to pay for products and services and to support personal (including home) banking and other financial activities.

10

VDE economically and efficiently fulfills the rights protection needs of electronic community members. Users of VDE will not require additional rights protection systems for different information highway products and rights

15 problems—nor will they be required to install and learn a new system for each new information highway application.

VDE provides a unified solution that allows all content creators, providers, and users to employ the same electronic

20 rights protection solution. Under authorized circumstances, the participants can freely exchange content and associated content control sets. This means that a user of VDE may, if allowed, use the same electronic system to work with different kinds of content having different sets of content control information. The

content and control information supplied by one group can be used by people who normally use content and control information supplied by a different group. VDE can allow content to be exchanged "universally" and users of an implementation of the present invention can interact electronically without fear of incompatibilities in content control, violation of rights, or the need to get, install, or learn a new content control system.

The VDE securely administers transactions that specify protection of rights. It can protect electronic rights including, for example:

- (a) the property rights of authors of electronic content,
- (b) the commercial rights of distributors of content,
- (c) the rights of any parties who facilitated the distribution of content,
- (d) the privacy rights of users of content,
- (e) the privacy rights of parties portrayed by stored and/or distributed content, and

- (f) any other rights regarding enforcement of electronic agreements.

5 VDE can enable a very broad variety of electronically enforced commercial and societal agreements. These agreements can include electronically implemented contracts, licenses, laws, regulations, and tax collection.

Contrast With Traditional Solutions

10 Traditional content control mechanisms often require users to purchase more electronic information than the user needs or desires. For example, infrequent users of shrink-wrapped software are required to purchase a program at the same price as frequent users, even though they may receive
15 much less value from their less frequent use. Traditional systems do not scale cost according to the extent or character of usage and traditional systems can not attract potential customers who find that a fixed price is too high. Systems using traditional mechanisms are also not normally particularly
20 secure. For example, shrink-wrapping does not prevent the constant illegal pirating of software once removed from either its physical or electronic package.

Traditional electronic information rights protection systems are often inflexible and inefficient and may cause a content provider to choose costly distribution channels that increase a product's price. In general these mechanisms restrict product pricing, configuration, and marketing flexibility. These compromises are the result of techniques for controlling information which cannot accommodate both different content models and content models which reflect the many, varied requirements, such as content delivery strategies, of the model participants. This can limit a provider's ability to deliver sufficient overall value to justify a given product's cost in the eyes of many potential users. VDE allows content providers and distributors to create applications and distribution networks that reflect content providers' and users' preferred business models. It offers users a uniquely cost effective and feature rich system that supports the ways providers *want* to distribute information and the ways users *want* to use such information. VDE supports content control models that ensure rights and allow content delivery strategies to be shaped for maximum commercial results.

Chain of Handling and Control

VDE can protect a collection of rights belonging to various parties having in rights in, or to, electronic information. This

information may be at one location or dispersed across (and/or moving between) multiple locations. The information may pass through a "chain" of distributors and a "chain" of users. Usage information may also be reported through one or more "chains" of parties. In general, VDE enables parties that (a) have rights in electronic information, and/or (b) act as direct or indirect agents for parties who have rights in electronic information, to ensure that the moving, accessing, modifying, or otherwise using of information can be securely controlled by rules regarding how, when, where, and by whom such activities can be performed.

VDE Applications and Software

VDE is a secure system for regulating electronic conduct and commerce. Regulation is ensured by control information put in place by one or more parties. These parties may include content providers, electronic hardware manufacturers, financial service providers, or electronic "infrastructure" companies such as cable or telecommunications companies. The control information implements "Rights Applications." Rights applications "run on" the "base software" of the preferred embodiment. This base software serves as a secure, flexible, general purpose foundation that can accommodate many different rights applications, that is, many different business models and their respective participant requirements.

A rights application under VDE is made up of special purpose pieces, each of which can correspond to one or more basic electronic processes needed for a rights protection environment. These processes can be combined together like building blocks to create electronic agreements that can protect the rights, and may enforce fulfillment of the obligations, of electronic information users and providers. One or more providers of electronic information can easily combine selected building blocks to create a rights application that is unique to a specific content distribution model. A group of these pieces can represent the capabilities needed to fulfill the agreement(s) between users and providers. These pieces accommodate many requirements of electronic commerce including:

- the distribution of permissions to use electronic information;
- the persistence of the control information and sets of control information managing these permissions;
- configurable control set information that can be selected by users for use with such information;

- data security and usage auditing of electronic information; and
- a secure system for currency, compensation and debit management.

5
10
15
20

For electronic commerce, a rights application, under the preferred embodiment of the present invention, can provide electronic enforcement of the business agreements between all participants. Since different groups of components can be put together for different applications, the present invention can provide electronic control information for a wide variety of different products and markets. This means the present invention can provide a "unified," efficient, secure, and cost-effective system for electronic commerce and data security. This allows VDE to serve as a single standard for electronic rights protection, data security, and electronic currency and banking.

20

In a VDE, the separation between a rights application and its foundation permits the efficient selection of sets of control information that are appropriate for each of many different types of applications and uses. These control sets can reflect both rights of electronic community members, as well as obligations

(such as providing a history of one's use of a product or paying taxes on one's electronic purchases). VDE flexibility allows its users to electronically implement and enforce common social and commercial ethics and practices. By providing a unified control system, the present invention supports a vast range of possible transaction related interests and concerns of individuals, communities, businesses, and governments. Due to its open design, VDE allows (normally under securely controlled circumstances) applications using technology independently created by users to be "added" to the system and used in conjunction with the foundation of the invention. In sum, VDE provides a system that can fairly reflect and enforce agreements among parties. It is a broad ranging and systematic solution that answers the pressing need for a secure, cost-effective, and fair electronic environment.

VDE Implementation

The preferred embodiment of the present invention includes various tools that enable system designers to directly insert VDE capabilities into their products. These tools include an Application Programmer's Interface ("API") and a Rights Permissioning and Management Language ("RPML"). The RPML provides comprehensive and detailed control over the use of the invention's features. VDE also includes certain user

interface subsystems for satisfying the needs of content providers, distributors, and users.

Information distributed using VDE may take many forms.

5 It may, for example, be "distributed" for use on an individual's own computer, that is the present invention can be used to provide security for locally stored data. Alternatively, VDE may be used with information that is dispersed by authors and/or publishers to one or more recipients. This information may take
10 many forms including: movies, audio recordings, games, electronic catalog shopping, multimedia, training materials, E-mail and personal documents, object oriented libraries, software programming resources, and reference/record keeping information resources (such as business, medical, legal,
15 scientific, governmental, and consumer databases).

Electronic rights protection provided by the present invention will also provide an important foundation for trusted and efficient home and commercial banking, electronic credit
20 processes, electronic purchasing, true or conditionally anonymous electronic cash, and EDI (Electronic Data Interchange). VDE provides important enhancements for improving data security in organizations by providing "smart"

transaction management features that can be far more effective than key and password based "go/no go" technology.

5 VDE normally employs an integration of cryptographic and other security technologies (e.g. encryption, digital signatures, etc.), with other technologies including: component, distributed, and event driven operating system technology, and related communications, object container, database, smart agent, smart card, and semiconductor design technologies.

10

I. Overview

A. VDE Solves Important Problems and Fills Critical Needs

15 The world is moving towards an integration of electronic information appliances. This interconnection of appliances provides a foundation for much greater electronic interaction and the evolution of electronic commerce. A variety of capabilities are required to implement an electronic commerce environment. VDE is the first system that provides many of these capabilities and therefore solves fundamental problems related to electronic
20 dissemination of information.

Electronic Content

- VDE allows electronic arrangements to be created involving two or more parties. These agreements can themselves comprise a collection of agreements between participants in a commercial value chain and/or a data security chain model for handling, auditing, reporting, and payment. It can provide efficient, reusable, modifiable, and consistent means for secure electronic content: distribution, usage control, usage payment, usage auditing, and usage reporting. Content may, for example, include:
- financial information such as electronic currency and credit;
 - commercially distributed electronic information such as reference databases, movies, games, and advertising; and
 - electronic properties produced by persons and organizations, such as documents, e-mail, and proprietary database information.

VDE enables an electronic commerce marketplace that supports differing, competitive business partnerships, agreements, and evolving overall business models.

5 The features of VDE allow it to function as the first
trusted electronic information control environment that can
conform to, and support, the bulk of conventional electronic
commerce and data security requirements. In particular, VDE
enables the participants in a business value chain model to
10 create an electronic version of traditional business agreement
terms and conditions and further enables these participants to
shape and evolve their electronic commerce models as they
believe appropriate to their business requirements.

15 VDE offers an architecture that avoids reflecting specific
distribution biases, administrative and control perspectives, and
content types. Instead, VDE provides a broad-spectrum,
fundamentally configurable and portable, electronic transaction
control, distributing, usage, auditing, reporting, and payment
20 operating environment. VDE is not limited to being an
application or application specific toolset that covers only a
limited subset of electronic interaction activities and
participants. Rather, VDE supports systems by which such
applications can be created, modified, and/or reused. As a result,

the present invention answers pressing, unsolved needs by offering a system that supports a standardized control environment which facilitates interoperability of electronic appliances, interoperability of content containers, and efficient
5 creation of electronic commerce applications and models through the use of a programmable, secure electronic transactions management foundation and reusable and extensible executable components. VDE can support a single electronic "world" within which most forms of electronic transaction activities can be
10 managed.

To answer the developing needs of rights owners and content providers and to provide a system that can accommodate the requirements and agreements of all parties that may be
15 involved in electronic business models (creators, distributors, administrators, users, credit providers, etc.), VDE supplies an efficient, largely transparent, low cost and sufficiently secure system (supporting both hardware/ software and software only models). VDE provides the widely varying secure control and
20 administration capabilities required for:

1. Different types of electronic content,
2. Differing electronic content delivery schemes,

3. Differing electronic content usage schemes,
4. Different content usage platforms, and
5. Differing content marketing and model strategies.

VDE may be combined with, or integrated into, many separate computers and/or other electronic appliances. These appliances typically include a secure subsystem that can enable control of content use such as displaying, encrypting, decrypting, printing, copying, saving, extracting, embedding, distributing, auditing usage, etc. The secure subsystem in the preferred embodiment comprises one or more "protected processing environments", one or more secure databases, and secure "component assemblies" and other items and processes that need to be kept secured. VDE can, for example, securely control electronic currency, payments, and/or credit management (including electronic credit and/or currency receipt, disbursement, encumbering, and/or allocation) using such a "secure subsystem."

VDE provides a secure, distributed electronic transaction management system for controlling the distribution and/or other usage of electronically provided and/or stored information. VDE

controls auditing and reporting of electronic content and/or
appliance usage. Users of VDE may include content creators
who apply content usage, usage reporting, and/or usage payment
related control information to electronic content and/or
5 appliances for users such as end-user organizations, individuals,
and content and/or appliance distributors. VDE also securely
supports the payment of money owed (including money owed for
content and/or appliance usage) by one or more parties to one or
more other parties, in the form of electronic credit and/or
10 currency.

Electronic appliances under control of VDE represent VDE
'nodes' that securely process and control; distributed electronic
information and/or appliance usage, control information
15 formulation, and related transactions. VDE can securely
manage the integration of control information provided by two or
more parties. As a result, VDE can construct an electronic
agreement between VDE participants that represent a
"negotiation" between, the control requirements of, two or more
20 parties and enacts terms and conditions of a resulting
agreement. VDE ensures the rights of each party to an
electronic agreement regarding a wide range of electronic
activities related to electronic information and/or appliance
usage.

Through use of VDE's control system, traditional content providers and users can create electronic relationships that reflect traditional, non-electronic relationships. They can shape and modify commercial relationships to accommodate the evolving needs of, and agreements among, themselves. VDE does not require electronic content providers and users to modify their business practices and personal preferences to conform to a metering and control application program that supports limited, largely fixed functionality. Furthermore, VDE permits participants to develop business models not feasible with non-electronic commerce, for example, involving detailed reporting of content usage information, large numbers of distinct transactions at hitherto infeasibly low price points, "pass-along" control information that is enforced without involvement or advance knowledge of the participants, etc.

The present invention allows content providers and users to formulate their transaction environment to accommodate:

- (1) desired content models, content control models, and content usage information pathways,
- (2) a complete range of electronic media and distribution means,

- 5
- (3) a broad range of pricing, payment, and auditing strategies,
- (4) very flexible privacy and/or reporting models,
- (5) practical and effective security architectures, and
- 10 (6) other administrative procedures that together with steps (1) through (5) can enable most "real world" electronic commerce and data security models, including models unique to the electronic world.

VDE's transaction management capabilities can enforce:

- 15 (1) privacy rights of users related to information regarding their usage of electronic information and/or appliances,
- 20 (2) societal policy such as laws that protect rights of content users or require the collection of taxes derived from electronic transaction revenue, and

- (3) the proprietary and/or other rights of parties related to ownership of, distribution of, and/or other commercial rights related to, electronic information.

5 VDE can support "real" commerce in an electronic form, that is the progressive creation of commercial relationships that form, over time, a network of interrelated agreements representing a value chain business model. This is achieved in part by enabling content control information to develop through
10 the interaction of (negotiation between) securely created and independently submitted sets of content and/or appliance control information. Different sets of content and/or appliance control information can be submitted by different parties in an electronic business value chain enabled by the present invention. These
15 parties create control information sets through the use of their respective VDE installations. Independently, securely deliverable, component based control information allows efficient interaction among control information sets supplied by different parties.

20

VDE permits multiple, separate electronic arrangements to be formed between subsets of parties in a VDE supported electronic value chain model. These multiple agreements together comprise a VDE value chain "extended" agreement.

VDE allows such constituent electronic agreements, and therefore overall VDE extended agreements, to evolve and reshape over time as additional VDE participants become involved in VDE content and/or appliance control information handling. VDE electronic agreements may also be extended as new control information is submitted by existing participants. With VDE, electronic commerce participants are free to structure and restructure their electronic commerce business activities and relationships. As a result, the present invention allows a competitive electronic commerce marketplace to develop since the use of VDE enables different, widely varying business models using the same or shared content.

A significant facet of the present invention's ability to broadly support electronic commerce is its ability to securely manage independently delivered VDE component objects containing control information (normally in the form of VDE objects containing one or more methods, data, or load module VDE components). This independently delivered control information can be integrated with senior and other pre-existing content control information to securely form derived control information using the negotiation mechanisms of the present invention. All requirements specified by this derived control information must be satisfied before VDE controlled content can

be accessed or otherwise used. This means that, for example, all load modules and any mediating data which are listed by the derived control information as required must be available and securely perform their required function. In combination with
5 other aspects of the present invention, securely, independently delivered control components allow electronic commerce participants to freely stipulate their business requirements and trade offs. As a result, much as with traditional, non-electronic commerce, the present invention allows electronic commerce
10 (through a progressive stipulation of various control requirements by VDE participants) to evolve into forms of business that are the most efficient, competitive and useful.

VDE provides capabilities that rationalize the support of
15 electronic commerce and electronic transaction management. This rationalization stems from the reusability of control structures and user interfaces for a wide variety of transaction management related activities. As a result, content usage control, data security, information auditing, and electronic
20 financial activities, can be supported with tools that are reusable, convenient, consistent, and familiar. In addition, a rational approach—a transaction/distribution control standard—allows all participants in VDE the same foundation set of hardware control and security, authoring, administration,

and management tools to support widely varying types of information, business market model, and/or personal objectives.

Employing VDE as a general purpose electronic

5 transaction/distribution control system allows users to maintain a single transaction management control arrangement on each of their computers, networks, communication nodes, and/or other electronic appliances. Such a general purpose system can serve the needs of many electronic transaction management

10 applications without requiring distinct, different installations for different purposes. As a result, users of VDE can avoid the confusion and expense and other inefficiencies of different, limited purpose transaction control applications for each different content and/or business model. For example, VDE

15 allows content creators to use the same VDE foundation control arrangement for both content authoring and for licensing content from other content creators for inclusion into their products or for other use. Clearinghouses, distributors, content creators, and other VDE users can all interact, both with the applications

20 running on their VDE installations, and with each other, in an entirely consistent manner, using and reusing (largely transparently) the same distributed tools, mechanisms, and consistent user interfaces, regardless of the type of VDE activity.

VDE prevents many forms of unauthorized use of electronic information, by controlling and auditing (and other administration of use) electronically stored and/or disseminated information. This includes, for example, commercially distributed content, electronic currency, electronic credit, business transactions (such as EDI), confidential communications, and the like. VDE can further be used to enable commercially provided electronic content to be made available to users in user defined portions, rather than constraining the user to use portions of content that were "predetermined" by a content creator and/or other provider for billing purposes.

VDE, for example, can employ:

- (1) Secure metering means for budgeting and/or auditing electronic content and/or appliance usage;
- (2) Secure flexible means for enabling compensation and/or billing rates for content and/or appliance usage, including electronic credit and/or currency mechanisms for payment means;

- (3) Secure distributed database means for storing control and usage related information (and employing validated compartmentalization and tagging schemes);

5

- (4) Secure electronic appliance control means;

10

- (5) A distributed, secure, "virtual black box" comprised of nodes located at every user (including VDE content container creators, other content providers, client users, and recipients of secure VDE content usage information) site. The nodes of said virtual black box normally include a secure subsystem having at least one secure hardware element (a semiconductor element or other hardware module for securely executing VDE control processes), said secure subsystems being distributed at nodes along a pathway of information storage, distribution, payment, usage, and/or auditing. In some embodiments, the functions of said hardware element, for certain or all nodes, may be performed by software, for example, in host processing environments of electronic appliances;

15

20

- 5
- (6) Encryption and decryption means;
- (7) Secure communications means employing authentication, digital signaturing, and encrypted transmissions. The secure subsystems at said user nodes utilize a protocol that establishes and authenticates each node's and/or participant's identity, and establishes one or more secure host-to-host encryption keys for communications between the secure subsystems; and
- 10
- (8) Secure control means that can allow each VDE installation to perform VDE content authoring (placing content into VDE containers with associated control information), content distribution, and content usage; as well as clearinghouse and other administrative and analysis activities employing content usage information.
- 15

20 VDE may be used to migrate most non-electronic, traditional information delivery models (including entertainment, reference materials, catalog shopping, etc.) into an adequately secure digital distribution and usage management

and payment context. The distribution and financial pathways managed by a VDE arrangement may include:

- 5 • content creator(s),
- distributor(s),
- redistributor(s),
- client administrator(s),
- client user(s),
- financial and/or other clearinghouse(s),
- 10 • and/or government agencies.

These distribution and financial pathways may also include:

- advertisers,
- 15 • market survey organizations, and/or
- other parties interested in the user usage of
information securely delivered and/or stored using
VDE.

20 Normally, participants in a VDE arrangement will employ the same secure VDE foundation. Alternate embodiments support VDE arrangements employing differing VDE foundations. Such alternate embodiments may employ procedures to ensure certain interoperability requirements are met.

Secure VDE hardware (also known as SPUs for Secure Processing Units), or VDE installations that use software to substitute for, or complement, said hardware (provided by Host Processing Environments (HPEs)), operate in conjunction with

5 secure communications, systems integration software, and distributed software control information and support structures, to achieve the electronic contract/rights protection environment of the present invention. Together, these VDE components

10 comprise a secure, virtual, distributed content and/or appliance control, auditing (and other administration), reporting, and payment environment. In some embodiments and where commercially acceptable, certain VDE participants, such as clearinghouses that normally maintain sufficiently physically

15 secure non-VDE processing environments, may be allowed to employ HPEs rather VDE hardware elements and interoperate, for example, with VDE end-users and content providers. VDE components together comprise a configurable, consistent, secure and "trusted" architecture for distributed, asynchronous control of electronic content and/or appliance usage. VDE supports a

20 "universe wide" environment for electronic content delivery, broad dissemination, usage reporting, and usage related payment activities.

VDE provides generalized configurability. This results, in part, from decomposition of generalized requirements for supporting electronic commerce and data security into a broad range of constituent "atomic" and higher level components (such as load modules, data elements, and methods) that may be variously aggregated together to form control methods for electronic commerce applications, commercial electronic agreements, and data security arrangements. VDE provides a secure operating environment employing VDE foundation elements along with secure independently deliverable VDE components that enable electronic commerce models and relationships to develop. VDE specifically supports the unfolding of distribution models in which content providers, over time, can expressly agree to, or allow, subsequent content providers and/or users to participate in shaping the control information for, and consequences of, use of electronic content and/or appliances. A very broad range of the functional attributes important for supporting simple to very complex electronic commerce and data security activities are supported by capabilities of the present invention. As a result, VDE supports most types of electronic information and/or appliance: usage control (including distribution), security, usage auditing, reporting, other administration, and payment arrangements.

VDE, in its preferred embodiment, employs object software technology and uses object technology to form "containers" for delivery of information that is (at least in part) encrypted or otherwise secured. These containers may contain electronic content products or other electronic information and some or all of their associated permissions (control) information. These container objects may be distributed along pathways involving content providers and/or content users. They may be securely moved among nodes of a Virtual Distribution Environment (VDE) arrangement, which nodes operate VDE foundation software and execute control methods to enact electronic information usage control and/or administration models. The containers delivered through use of the preferred embodiment of the present invention may be employed both for distributing VDE control instructions (information) and/or to encapsulate and electronically distribute content that has been at least partially secured.

Content providers who employ the present invention may include, for example, software application and game publishers, database publishers, cable, television, and radio broadcasters, electronic shopping vendors, and distributors of information in electronic document, book, periodical, e-mail and/or other forms. Corporations, government agencies, and/or individual

“end-users” who act as storers of, and/or distributors of, electronic information, may also be VDE content providers (in a restricted model, a user provides content only to himself and employs VDE to secure his own confidential information against unauthorized use by other parties). Electronic information may include proprietary and/or confidential information for personal or internal organization use, as well as information, such as software applications, documents, entertainment materials, and/or reference information, which may be provided to other parties. Distribution may be by, for example, physical media delivery, broadcast and/or telecommunication means, and in the form of “static” files and/or streams of data. VDE may also be used, for example, for multi-site “real-time” interaction such as teleconferencing, interactive games, or on-line bulletin boards, where restrictions on, and/or auditing of, the use of all or portions of communicated information is enforced.

VDE provides important mechanisms for both enforcing commercial agreements and enabling the protection of privacy rights. VDE can securely deliver information from one party to another concerning the use of commercially distributed electronic content. Even if parties are separated by several “steps” in a chain (pathway) of handling for such content usage information, such information is protected by VDE through encryption and/or

other secure processing. Because of that protection, the accuracy of such information is guaranteed by VDE, and the information can be trusted by all parties to whom it is delivered.

Furthermore, VDE guarantees that all parties can trust that
5 such information cannot be received by anyone other than the intended, authorized, party(ies) because it is encrypted such that only an authorized party, or her agents, can decrypt it. Such information may also be derived through a secure VDE process at a previous pathway-of-handling location to produce secure
10 VDE reporting information that is then communicated securely to its intended recipient's VDE secure subsystem. Because VDE can deliver such information securely, parties to an electronic agreement need not trust the accuracy of commercial usage and/or other information delivered through means other than
15 those under control of VDE.

VDE participants in a commercial value chain can be "commercially" confident (that is, sufficiently confident for commercial purposes) that the direct (constituent) and/or
20 "extended" electronic agreements they entered into through the use of VDE can be enforced reliably. These agreements may have both "dynamic" transaction management related aspects, such as content usage control information enforced through budgeting, metering, and/or reporting of electronic information

and/or appliance use, and/or they may include "static" electronic assertions, such as an end-user using the system to assert his or her agreement to pay for services, not to pass to unauthorized parties electronic information derived from usage of content or systems, and/or agreeing to observe copyright laws. Not only can electronically reported transaction related information be trusted under the present invention, but payment may be automated by the passing of payment tokens through a pathway of payment (which may or may not be the same as a pathway for reporting).

Such payment can be contained within a VDE container created automatically by a VDE installation in response to control information (located, in the preferred embodiment, in one or more permissions records) stipulating the "withdrawal" of credit or electronic currency (such as tokens) from an electronic account (for example, an account securely maintained by a user's VDE installation secure subsystem) based upon usage of VDE controlled electronic content and/or appliances (such as governments, financial credit providers, and users).

VDE allows the needs of electronic commerce participants to be served and it can bind such participants together in a universe wide, trusted commercial network that can be secure enough to support very large amounts of commerce. VDE's security and metering secure subsystem core will be present at

all physical locations where VDE related content is (a) assigned
usage related control information (rules and mediating data),
and/or (b) used. This core can perform security and auditing
functions (including metering) that operate within a "virtual
5 black box," a collection of distributed, very secure VDE related
hardware instances that are interconnected by secured
information exchange (for example, telecommunication)
processes and distributed database means. VDE further
includes highly configurable transaction operating system
10 technology, one or more associated libraries of load modules
along with affiliated data, VDE related administration, data
preparation, and analysis applications, as well as system
software designed to enable VDE integration into host
environments and applications. VDE's usage control
15 information, for example, provide for property content and/or
appliance related: usage authorization, usage auditing (which
may include audit reduction), usage billing, usage payment,
privacy filtering, reporting, and security related communication
and encryption techniques.

20

VDE extensively employs methods in the form of software
objects to augment configurability, portability, and security of
the VDE environment. It also employs a software object
architecture for VDE content containers that carries protected

content and may also carry both freely available information (e.g., summary, table of contents) and secured content control information which ensures the performance of control information. Content control information governs content usage according to criteria set by holders of rights to an object's contents and/or according to parties who otherwise have rights associated with distributing such content (such as governments, financial credit providers, and users).

10 In part, security is enhanced by object methods employed by the present invention because the encryption schemes used to protect an object can efficiently be further used to protect the associated content control information (software control information and relevant data) from modification. Said object techniques also enhance portability between various computer and/or other appliance environments because electronic information in the form of content can be inserted along with (for example, in the same object container as) content control information (for said content) to produce a "published" object.

15

20 As a result, various portions of said control information may be specifically adapted for different environments, such as for diverse computer platforms and operating systems, and said various porticns may all be carried by a VDE container.

An objective of VDE is supporting a transaction/distribution control standard. Development of such a standard has many obstacles, given the security requirements and related hardware and communications issues, widely
5 differing environments, information types, types of information usage, business and/or data security goals, varieties of participants, and properties of delivered information. A significant feature of VDE accommodates the many, varying distribution and other transaction variables by, in part,
10 decomposing electronic commerce and data security functions into generalized capability modules executable within a secure hardware SPU and/or corresponding software subsystem and further allowing extensive flexibility in assembling, modifying, and/or replacing, such modules (e.g. load modules and/or
15 methods) in applications run on a VDE installation foundation. This configurability and reconfigurability allows electronic commerce and data security participants to reflect their priorities and requirements through a process of iteratively shaping an evolving extended electronic agreement (electronic
20 control model). This shaping can occur as content control information passes from one VDE participant to another and to the extent allowed by "in place" content control information. This process allows users of VDE to recast existing control

information and/or add new control information as necessary
(including the elimination of no longer required elements).

5 VDE supports trusted (sufficiently secure) electronic
information distribution and usage control models for both
commercial electronic content distribution and data security
applications. It can be configured to meet the diverse
requirements of a network of interrelated participants that may
include content creators, content distributors, client
10 administrators, end users, and/or clearinghouses and/or other
content usage information users. These parties may constitute a
network of participants involved in simple to complex electronic
content dissemination, usage control, usage reporting, and/or
usage payment. Disseminated content may include both
15 originally provided and VDE generated information (such as
content usage information) and content control information may
persist through both chains (one or more pathways) of content
and content control information handling, as well as the direct
usage of content. The configurability provided by the present
20 invention is particularly critical for supporting electronic
commerce, that is enabling businesses to create relationships
and evolve strategies that offer competitive value. Electronic
commerce tools that are not inherently configurable and
interoperable will ultimately fail to produce products (and

services) that meet both basic requirements and evolving needs of most commerce applications.

5 VDE's fundamental configurability will allow a broad range of competitive electronic commerce business models to flourish. It allows business models to be shaped to maximize revenues sources, end-user product value, and operating efficiencies. VDE can be employed to support multiple, differing models, take advantage of new revenue opportunities, and
10 deliver product configurations most desired by users. Electronic commerce technologies that do not, as the present invention does:

- support a broad range of possible, complementary revenue activities,
 - 15 • offer a flexible array of content usage features most desired by customers, and
 - exploit opportunities for operating efficiencies,
- will result in products that are often intrinsically more costly and less appealing and therefore less competitive in the
20 marketplace.

Some of the key factors contributing to the configurability intrinsic to the present invention include:

- 5 (a) integration into the fundamental control environment of a broad range of electronic appliances through portable API and programming language tools that efficiently support merging of control and auditing capabilities in nearly any electronic appliance environment while maintaining overall system security;
- 10 (b) modular data structures;
- (c) generic content model;
- 15 (d) general modularity and independence of foundation architectural components;
- (e) modular security structures;
- (f) variable length and multiple branching chains of control; and
- 20 (g) independent, modular control structures in the form of executable load modules that can be maintained in one or more libraries, and assembled into control methods and models, and where such model control

schemes can "evolve" as control information passes through the VDE installations of participants of a pathway of VDE content control information handling.

5

Because of the breadth of issues resolved by the present invention, it can provide the emerging "electronic highway" with a single transaction/distribution control system that can, for a very broad range of commercial and data security models, ensure
10 against unauthorized use of confidential and/or proprietary information and commercial electronic transactions. VDE's electronic transaction management mechanisms can enforce the electronic rights and agreements of all parties participating in widely varying business and data security models, and this can
15 be efficiently achieved through a single VDE implementation within each VDE participant's electronic appliance. VDE supports widely varying business and/or data security models that can involve a broad range of participants at various "levels" of VDE content and/or content control information pathways of
20 handling. Different content control and/or auditing models and agreements may be available on the same VDE installation. These models and agreements may control content in relationship to, for example, VDE installations and/or users in general; certain specific users, installations, classes and/or other

groupings of installations and/or users; as well as to electronic content generally on a given installation, to specific properties, property portions, classes and/or other groupings of content.

5 Distribution using VDE may package both the electronic content and control information into the same VDE container, and/or may involve the delivery to an end-user site of different pieces of the same VDE managed property from plural separate remote locations and/or in plural separate VDE content
10 containers and/or employing plural different delivery means. Content control information may be partially or fully delivered separately from its associated content to a user VDE installation in one or more VDE administrative objects. Portions of said control information may be delivered from one or more sources.
15 Control information may also be available for use by access from a user's VDE installation secure sub-system to one or more remote VDE secure sub-systems and/or VDE compatible, certified secure remote locations. VDE control processes such as metering, budgeting, decrypting and/or fingerprinting, may as
20 relates to a certain user content usage activity, be performed in a user's local VDE installation secure subsystem, or said processes may be divided amongst plural secure subsystems which may be located in the same user VDE installations and/or in a network server and in the user installation. For example, a local VDE

installation may perform decryption and save any, or all of, usage metering information related to content and/or electronic appliance usage at such user installation could be performed at the server employing secure (e.g., encrypted) communications
5 between said secure subsystems. Said server location may also be used for near real time, frequent, or more periodic secure receipt of content usage information from said user installation, with, for example, metered information being maintained only temporarily at a local user installation.

10

Delivery means for VDE managed content may include electronic data storage means such as optical disks for delivering one portion of said information and broadcasting and/or telecommunicating means for other portions of said information.

15 Electronic data storage means may include magnetic media, optical media, combined magneto-optical systems, flash RAM memory, bubble memory, and/or other memory storage means such as huge capacity optical storage systems employing holographic, frequency, and/or polarity data storage techniques.

20 Data storage means may also employ layered disc techniques, such as the use of generally transparent and/or translucent materials that pass light through layers of data carrying discs which themselves are physically packaged together as one

thicker disc. Data carrying locations on such discs may be, at least in part, opaque.

5 VDE supports a general purpose foundation for secure transaction management, including usage control, auditing, reporting, and/or payment. This general purpose foundation is called "VDE Functions" ("VDEFs"). VDE also supports a collection of "atomic" application elements (e.g., load modules) that can be selectively aggregated together to form various
10 VDEF capabilities called control methods and which serve as VDEF applications and operating system functions. When a host operating environment of an electronic appliance includes VDEF capabilities, it is called a "Rights Operating System" (ROS). VDEF load modules, associated data, and methods form a body of
15 information that for the purposes of the present invention are called "control information." VDEF control information may be specifically associated with one or more pieces of electronic content and/or it may be employed as a general component of the operating system capabilities of a VDE installation.

20

VDEF transaction control elements reflect and enact content specific and/or more generalized administrative (for example, general operating system) control information. VDEF capabilities which can generally take the form of applications

(application models) that have more or less configurability which can be shaped by VDE participants, through the use, for example, of VDE templates, to employ specific capabilities, along, for example, with capability parameter data to reflect the

5 elements of one or more express electronic agreements between VDE participants in regards to the use of electronic content such as commercially distributed products. These control capabilities manage the use of, and/or auditing of use of, electronic content, as well as reporting information based upon content use, and any

10 payment for said use. VDEF capabilities may "evolve" to reflect the requirements of one or more successive parties who receive or otherwise contribute to a given set of control information. Frequently, for a VDE application for a given content model (such as distribution of entertainment on CD-ROM, content

15 delivery from an Internet repository, or electronic catalog shopping and advertising, or some combination of the above) participants would be able to securely select from amongst available, alternative control methods and apply related parameter data, wherein such selection of control method and/or

20 submission of data would constitute their "contribution" of control information. Alternatively, or in addition, certain control methods that have been expressly certified as securely interoperable and compatible with said application may be independently submitted by a participant as part of such a

contribution. In the most general example, a generally certified load module (certified for a given VDE arrangement and/or content class) may be used with many or any VDE application that operates in nodes of said arrangement. These parties, to the extent they are allowed, can independently and securely add, delete, and/or otherwise modify the specification of load modules and methods, as well as add, delete or otherwise modify related information.

10 Normally the party who creates a VDE content container defines the general nature of the VDEF capabilities that will and/or may apply to certain electronic information. A VDE content container is an object that contains both content (for example, commercially distributed electronic information products such as computer software programs, movies, electronic publications or reference materials, etc.) and certain control information related to the use of the object's content. A creating party may make a VDE container available to other parties. Control information delivered by, and/or otherwise available for use with, VDE content containers comprise (for commercial content distribution purposes) VDEF control capabilities (and any associated parameter data) for electronic content. These capabilities may constitute one or more "proposed" electronic agreements (and/or agreement functions available for selection

and/or use with parameter data) that manage the use and/or the consequences of use of such content and which can enact the terms and conditions of agreements involving multiple parties and their various rights and obligations.

5

A VDE electronic agreement may be explicit, through a user interface acceptance by one or more parties, for example by a "junior" party who has received control information from a "senior" party, or it may be a process amongst equal parties who individually assert their agreement. Agreement may also result from an automated electronic process during which terms and conditions are "evaluated" by certain VDE participant control information that assesses whether certain other electronic terms and conditions attached to content and/or submitted by another party are acceptable (do not violate acceptable control information criteria). Such an evaluation process may be quite simple, for example a comparison to ensure compatibility between a portion of, or all senior, control terms and conditions in a table of terms and conditions and the submitted control information of a subsequent participant in a pathway of content control information handling, or it may be a more elaborate process that evaluates the potential outcome of, and/or implements a negotiation process between, two or more sets of control information submitted by two or more parties. VDE also

10

15

20

accommodates a semi-automated process during which one or more VDE participants directly, through user interface means, resolve "disagreements" between control information sets by accepting and/or proposing certain control information that may
5 be acceptable to control information representing one or more other parties interests and/or responds to certain user interface queries for selection of certain alternative choices and/or for certain parameter information, the responses being adopted if acceptable to applicable senior control information.

10

When another party (other than the first applier of rules), perhaps through a negotiation process, accepts, and/or adds to and/or otherwise modifies, "in place" content control information, a VDE agreement between two or more parties related to the use
15 of such electronic content may be created (so long as any modifications are consistent with senior control information). Acceptance of terms and conditions related to certain electronic content may be direct and express, or it may be implicit as a result of use of content (depending, for example, on legal
20 requirements, previous exposure to such terms and conditions, and requirements of in place control information).

VDEF capabilities may be employed, and a VDE agreement may be entered into, by a plurality of parties without

the VDEF capabilities being directly associated with the controlling of certain, specific electronic information. For example, certain one or more VDEF capabilities may be present at a VDE installation, and certain VDE agreements may have
5 been entered into during the registration process for a content distribution application, to be used by such installation for securely controlling VDE content usage, auditing, reporting and/or payment. Similarly, a specific VDE participant may enter into a VDE user agreement with a VDE content or electronic
10 appliance provider when the user and/or her appliance register with such provider as a VDE installation and/or user. In such events, VDEF in place control information available to the user VDE installation may require that certain VDEF methods are employed, for example in a certain sequence. in order to be able
15 to use all and/or certain classes, of electronic content and/or VDE applications.

VDE ensures that certain prerequisites necessary for a given transaction to occur are met. This includes the secure
20 execution of any required load modules and the availability of any required, associated data. For example, required load modules and data (e.g. in the form of a method) might specify that sufficient credit from an authorized source must be confirmed as available. It might further require certain one or

more load modules execute as processes at an appropriate time to ensure that such credit will be used in order to pay for user use of the content. A certain content provider might, for example, require metering the number of copies made for distribution to employees of a given software program (a portion of the program might be maintained in encrypted form and require the presence of a VDE installation to run). This would require the execution of a metering method for copying of the property each time a copy was made for another employee. This same provider might also charge fees based on the total number of different properties licensed from them by the user and a metering history of their licensing of properties might be required to maintain this information.

VDE provides organization, community, and/or universe wide secure environments whose integrity is assured by processes securely controlled in VDE participant user installations (nodes). VDE installations, in the preferred embodiment, may include both software and tamper resistant hardware semiconductor elements. Such a semiconductor arrangement comprises, at least in part, special purpose circuitry that has been designed to protect against tampering with, or unauthorized observation of, the information and functions used in performing the VDE's control functions. The special purpose

secure circuitry provided by the present invention includes at least one of: a dedicated semiconductor arrangement known as a Secure Processing Unit (SPU) and/or a standard microprocessor, microcontroller, and/or other processing logic that accommodates the requirements of the present invention and functions as an SPU. VDE's secure hardware may be found incorporated into, for example, a fax/modem chip or chip pack, I/O controller, video display controller, and/or other available digital processing arrangements. It is anticipated that portions of the present invention's VDE secure hardware capabilities may ultimately be standard design elements of central processing units (CPUs) for computers and various other electronic devices.

Designing VDE capabilities into one or more standard microprocessor, microcontroller and/or other digital processing components may materially reduce VDE related hardware costs by employing the same hardware resources for both the transaction management uses contemplated by the present invention and for other, host electronic appliance functions. This means that a VDE SPU can employ (share) circuitry elements of a "standard" CPU. For example, if a "standard" processor can operate in protected mode and can execute VDE related instructions as a protected activity, then such an embodiment may provide sufficient hardware security for a variety of

applications and the expense of a special purpose processor might be avoided. Under one preferred embodiment of the present invention, certain memory (e.g., RAM, ROM, NVRAM) is maintained during VDE related instruction processing in a protected mode (for example, as supported by protected mode microprocessors). This memory is located in the same package as the processing logic (e.g. processor). Desirably, the packaging and memory of such a processor would be designed using security techniques that enhance its resistance to tampering.

10

The degree of overall security of the VDE system is primarily dependent on the degree of tamper resistance and concealment of VDE control process execution and related data storage activities. Employing special purpose semiconductor packaging techniques can significantly contribute to the degree of security. Concealment and tamper-resistance in semiconductor memory (e.g., RAM, ROM, NVRAM) can be achieved, in part, by employing such memory within an SPU package, by encrypting data before it is sent to external memory (such as an external RAM package) and decrypting encrypted data within the CPU/RAM package before it is executed. This process is used for important VDE related data when such data is stored on unprotected media, for example, standard host storage, such as random access memory, mass storage, etc. In

15

20

that event, a VDE SPU would encrypt data that results from a secure VDE execution before such data was stored in external memory.

5 **Summary of Some Important Features Provided by VDE in Accordance With the Present Invention**

VDE employs a variety of capabilities that serve as a foundation for a general purpose, sufficiently secure distributed electronic commerce solution. VDE enables an electronic commerce marketplace that supports divergent, competitive
10 business partnerships, agreements, and evolving overall business models. For example, VDE includes features that:

“sufficiently” impede unauthorized and/or
15 uncompensated use of electronic information and/or appliances through the use of secure communication, storage, and transaction management technologies. VDE supports a model wide, distributed security implementation which
20 creates a single secure “virtual” transaction processing and information storage environment. VDE enables distributed VDE installations to securely store and communicate information and remotely control the execution processes and the

character of use of electronic information at other
VDE installations and in a wide variety of ways;

- 5 • support low-cost, efficient, and effective security
architectures for transaction control, auditing,
reporting, and related communications and
information storage. VDE may employ tagging
related security techniques, the time-ageing of
10 encryption keys, the compartmentalization of both
stored control information (including differentially
tagging such stored information to ensure against
substitution and tampering) and distributed content
(to, for many content applications, employ one or
more content encryption keys that are unique to the
15 specific VDE installation and/or user), private key
techniques such as triple DES to encrypt content,
public key techniques such as RSA to protect
communications and to provide the benefits of
digital signature and authentication to securely bind
20 together the nodes of a VDE arrangement, secure
processing of important transaction management
executable code, and a combining of a small amount
of highly secure, hardware protected storage space
with a much larger "exposed" mass media storage

space storing secured (normally encrypted and tagged) control and audit information. VDE employs special purpose hardware distributed throughout some or all locations of a VDE implementation: a) said hardware controlling important elements of: content preparation (such as causing such content to be placed in a VDE content container and associating content control information with said content), content and/or electronic appliance usage auditing, content usage analysis, as well as content usage control; and b) said hardware having been designed to securely handle processing load module control activities, wherein said control processing activities may involve a sequence of required control factors;

- support dynamic user selection of information subsets of a VDE electronic information product (VDE controlled content). This contrasts with the constraints of having to use a few high level individual, pre-defined content provider information increments such as being required to select a whole information product or product section in order to acquire or otherwise use a portion of such product or

section. VDE supports metering and usage control over a variety of increments (including "atomic" increments, and combinations of different increment types) that are selected ad hoc by a user and represent a collection of pre-identified one or more increments (such as one or more blocks of a preidentified nature, e.g., bytes, images, logically related blocks) that form a generally arbitrary, but logical to a user, content "deliverable." VDE control information (including budgeting, pricing and metering) can be configured so that it can specifically apply, as appropriate, to ad hoc selection of different, unanticipated variable user selected aggregations of information increments and pricing levels can be, at least in part, based on quantities and/or nature of mixed increment selections (for example, a certain quantity of certain text could mean associated images might be discounted by 15%; a greater quantity of text in the "mixed" increment selection might mean the images are discounted 20%). Such user selected aggregated information increments can reflect the actual requirements of a user for information and is more flexible than being limited to a single, or a few, high

level, (e.g. product, document, database record)
predetermined increments. Such high level
increments may include quantities of information
not desired by the user and as a result be more
5 costly than the subset of information needed by the
user if such a subset was available. In sum, the
present invention allows information contained in
electronic information products to be supplied
according to user specification. Tailoring to user
10 specification allows the present invention to provide
the greatest value to users, which in turn will
generate the greatest amount of electronic commerce
activity. The user, for example, would be able to
define an aggregation of content derived from
15 various portions of an available content product, but
which, as a deliverable for use by the user, is an
entirely unique aggregated increment. The user
may, for example, select certain numbers of bytes of
information from various portions of an information
20 product, such as a reference work, and copy them to
disc in unencrypted form and be billed based on
total number of bytes plus a surcharge on the
number of "articles" that provided the bytes. A
content provider might reasonably charge less for

such a user defined information increment since the user does not require all of the content from all of the articles that contained desired information. This process of defining a user desired information increment may involve artificial intelligence database search tools that contribute to the location of the most relevant portions of information from an information product and cause the automatic display to the user of information describing search criteria hits for user selection or the automatic extraction and delivery of such portions to the user. VDE further supports a wide variety of predefined increment types including:

- bytes,
- images,
- content over time for audio or video, or any other increment that can be identified by content provider data mapping efforts, such as:
 - sentences,
 - paragraphs,
 - articles,
 - database records, and
 - byte offsets representing increments of logically related information.

VDE supports as many simultaneous predefined increment types as may be practical for a given type of content and business model.

- 5 • securely store at a user's site potentially highly detailed information reflective of a user's usage of a variety of different content segment types and employing both inexpensive "exposed" host mass storage for maintaining detailed information in the form of encrypted data and maintaining summary information for security testing in highly secure special purpose VDE installation nonvolatile memory (if available).
- 10
- 15 • support trusted chain of handling capabilities for pathways of distributed electronic information and/or for content usage related information. Such chains may extend, for example, from a content creator, to a distributor, a redistributor, a client user, and then may provide a pathway for securely reporting the same and/or differing usage information to one or more auditors, such as to one or more independent clearinghouses and then back to the content providers, including content creators.
- 20

5 The same and/or different pathways employed for
certain content handling, and related content control
information and reporting information handling,
may also be employed as one or more pathways for
electronic payment handling (payment is
characterized in the present invention as
administrative content) for electronic content and/or
appliance usage. These pathways are used for
conveyance of all or portions of content, and/or
10 content related control information. Content
creators and other providers can specify the
pathways that, partially or fully, must be used to
disseminate commercially distributed property
content, content control information, payment
15 administrative content, and/or associated usage
reporting information. Control information specified
by content providers may also specify which specific
parties must or may (including, for example, a group
of eligible parties from which a selection may be
20 made) handle conveyed information. It may also
specify what transmission means (for example
telecommunication carriers or media types) and
transmission hubs must or may be used.

- support flexible auditing mechanisms, such as employing "bitmap meters," that achieve a high degree of efficiency of operation and throughput and allow, in a practical manner, the retention and ready recall of information related to previous usage activities and related patterns. This flexibility is adaptable to a wide variety of billing and security control strategies such as:
 - upgrade pricing (e.g. suite purchases),
 - pricing discounts (including quantity discounts),
 - billing related time duration variables such as discounting new purchases based on the timing of past purchases, and
 - security budgets based on quantity of different, logically related units of electronic information used over an interval of time.

Use of bitmap meters (including "regular" and "wide" bitmap meters) to record usage and/or purchase of information, in conjunction with other elements of the preferred embodiment of the present invention, uniquely supports efficient maintenance of usage history for: (a) rental, (b) flat fee licensing

or purchase, (c) licensing or purchase discounts based upon historical usage variables, and (d) reporting to users in a manner enabling users to determine whether a certain item was acquired, or
5 acquired within a certain time period (without requiring the use of conventional database mechanisms, which are highly inefficient for these applications). Bitmap meter methods record activities associated with electronic appliances,
10 properties, objects, or portions thereof, and/or administrative activities that are independent of specific properties, objects, etc., performed by a user and/or electronic appliance such that a content and/or appliance provider and/or controller of an
15 administrative activity can determine whether a certain activity has occurred at some point, or during a certain period, in the past (for example, certain use of a commercial electronic content product and/or appliance). Such determinations can
20 then be used as part of pricing and/or control strategies of a content and/or appliance provider, and/or controller of an administrative activity. For example, the content provider may choose to charge only once for access to a portion of a property,

regardless of the number of times that portion of the property is accessed by a user.

- 5 • support "launchable" content, that is content that
can be provided by a content provider to an
end-user, who can then copy or pass along the
content to other end-user parties without requiring
the direct participation of a content provider to
register and/or otherwise initialize the content for
10 use. This content goes "out of (the traditional
distribution) channel" in the form of a "traveling
object." Traveling objects are containers that
securely carry at least some permissions information
and/or methods that are required for their use (such
15 methods need not be carried by traveling objects if
the required methods will be available at, or directly
available to, a destination VDE installation).
Certain travelling objects may be used at some or all
VDE installations of a given VDE arrangement since
20 they can make available the content control
information necessary for content use without
requiring the involvement of a commercial VDE
value chain participant or data security
administrator (e.g. a control officer or network

administrator). As long as traveling object control information requirements are available at the user VDE installation secure subsystem (such as the presence of a sufficient quantity of financial credit from an authorized credit provider), at least some travelling object content may be used by a receiving party without the need to establish a connection with a remote VDE authority (until, for example, budgets are exhausted or a time content usage reporting interval has occurred). Traveling objects can travel "out-of-channel," allowing, for example, a user to give a copy of a traveling object whose content is a software program, a movie or a game, to a neighbor, the neighbor being able to use the traveling object if appropriate credit (e.g. an electronic clearinghouse account from a clearinghouse such as VISA or AT&T) is available. Similarly, electronic information that is generally available on an Internet, or a similar network, repository might be provided in the form of a traveling object that can be downloaded and subsequently copied by the initial downloader and then passed along to other parties who may pass the object on to additional parties.

- 5 • provide very flexible and extensible user identification according to individuals, installations, by groups such as classes, and by function and hierarchical identification employing a hierarchy of levels of client identification (for example, client organization ID, client department ID, client network ID, client project ID, and client employee ID, or any appropriate subset of the above).
- 10 • provide a general purpose, secure, component based content control and distribution system that functions as a foundation transaction operating system environment that employs executable code pieces crafted for transaction control and auditing.
- 15 These code pieces can be reused to optimize efficiency in creation and operation of trusted, distributed transaction management arrangements. VDE supports providing such executable code in the form of "atomic" load modules and associated data.
- 20 Many such load modules are inherently configurable, aggregatable, portable, and extensible and singularly, or in combination (along with associated data), run as control methods under the VDE transaction operating environment. VDE can

5 satisfy the requirements of widely differing
electronic commerce and data security applications
by, in part, employing this general purpose
transaction management foundation to securely
process VDE transaction related control methods.
Control methods are created primarily through the
use of one or more of said executable, reusable load
module code pieces (normally in the form of
executable object components) and associated data.
10 The component nature of control methods allows the
present invention to efficiently operate as a highly
configurable content control system. Under the
present invention, content control models can be
iteratively and asynchronously shaped, and
15 otherwise updated to accommodate the needs of
VDE participants to the extent that such shaping
and otherwise updating conforms to constraints
applied by a VDE application, if any (e.g., whether
new component assemblies are accepted and, if so,
20 what certification requirements exist for such
component assemblies or whether any or certain
participants may shape any or certain control
information by selection amongst optional control
information (permissions record) control methods.

This iterative (or concurrent) multiple participant process occurs as a result of the submission and use of secure, control information components (executable code such as load modules and/or methods, and/or associated data). These components may be contributed independently by secure communication between each control information influencing VDE participant's VDE installation and may require certification for use with a given application, where such certification was provided by a certification service manager for the VDE arrangement who ensures secure interoperability and/or reliability (e.g., bug control resulting from interaction) between appliances and submitted control methods. The transaction management control functions of a VDE electronic appliance transaction operating environment interact with non-secure transaction management operating system functions to properly direct transaction processes and data related to electronic information security, usage control, auditing, and usage reporting. VDE provides the capability to manages resources related to secure VDE content

and/or appliance control information execution and data storage.

- 5 ● facilitate creation of application and/or system
functionality under VDE and to facilitate integration
into electronic appliance environments of load
modules and methods created under the present
invention. To achieve this, VDE employs an
Application Programmer's Interface (API) and/or a
10 transaction operating system (such as a ROS)
programming language with incorporated functions,
both of which support the use of capabilities and can
be used to efficiently and tightly integrate VDE
functionality into commercial and user applications.
15
- 20 ● support user interaction through: (a) "Pop-Up"
applications which, for example, provide messages to
users and enable users to take specific actions such
as approving a transaction, (b) stand-alone VDE
applications that provide administrative
environments for user activities such as: end-user
preference specifications for limiting the price per
transaction, unit of time, and/or session, for

accessing history information concerning previous transactions, for reviewing financial information such as budgets, expenditures (e.g. detailed and/or summary) and usage analysis information, and (c)

5 VDE aware applications which, as a result of the use of a VDE API and/or a transaction management (for example, ROS based) programming language embeds VDE "awareness" into commercial or internal software (application programs, games, etc.)

10 so that VDE user control information and services are seamlessly integrated into such software and can be directly accessed by a user since the underlying functionality has been integrated into the commercial software's native design. For

15 example, in a VDE aware word processor application, a user may be able to "print" a document into a VDE content container object, applying specific control information by selecting from amongst a series of different menu templates

20 for different purposes (for example, a confidential memo template for internal organization purposes may restrict the ability to "keep," that is to make an electronic copy of the memo).

- employ "templates" to ease the process of configuring capabilities of the present invention as they relate to specific industries or businesses. Templates are applications or application add-ons under the present invention. Templates support the efficient specification and/or manipulation of criteria related to specific content types, distribution approaches, pricing mechanisms, user interactions with content and/or administrative activities, and/or the like.
- Given the very large range of capabilities and configurations supported by the present invention, reducing the range of configuration opportunities to a manageable subset particularly appropriate for a given business model allows the full configurable power of the present invention to be easily employed by "typical" users who would be otherwise burdened with complex programming and/or configuration design responsibilities template applications can also help ensure that VDE related processes are secure and optimally bug free by reducing the risks associated with the contribution of independently developed load modules, including unpredictable aspects of code interaction between independent modules and applications, as well as security risks

associated with possible presence of viruses in such modules. VDE, through the use of templates, reduces typical user configuration responsibilities to an appropriately focused set of activities including selection of method types (e.g. functionality) through menu choices such as multiple choice, icon selection, and/or prompting for method parameter data (such as identification information, prices, budget limits, dates, periods of time, access rights to specific content, etc.) that supply appropriate and/or necessary data for control information purposes. By limiting the typical (non-programming) user to a limited subset of configuration activities whose general configuration environment (template) has been preset to reflect general requirements corresponding to that user, or a content or other business model can very substantially limit difficulties associated with content containerization (including placing initial control information on content), distribution, client administration, electronic agreement implementation, end-user interaction, and clearinghouse activities, including associated interoperability problems (such as conflicts resulting from security, operating system,

and/or certification incompatibilities). Use of appropriate VDE templates can assure users that their activities related to content VDE containerization, contribution of other control information, communications, encryption techniques and/or keys, etc. will be in compliance with specifications for their distributed VDE arrangement. VDE templates constitute preset configurations that can normally be reconfigurable to allow for new and/or modified templates that reflect adaptation into new industries as they evolve or to reflect the evolution or other change of an existing industry. For example, the template concept may be used to provide individual, overall frameworks for organizations and individuals that create, modify, market, distribute, consume, and/or otherwise use movies, audio recordings and live performances, magazines, telephony based retail sales, catalogs, computer software, information data bases, multimedia, commercial communications, advertisements, market surveys, infomercials, games, CAD/CAM services for numerically controlled machines, and the like. As the context surrounding these templates changes or evolves,

template applications provided under the present invention may be modified to meet these changes for broad use, or for more focused activities. A given VDE participant may have a plurality of templates available for different tasks. A party that places content in its initial VDE container may have a variety of different, configurable templates depending on the type of content and/or business model related to the content. An end-user may have different configurable templates that can be applied to different document types (e-mail, secure internal documents, database records, etc.) and/or subsets of users (applying differing general sets of control information to different bodies of users, for example, selecting a list of users who may, under certain preset criteria, use a certain document). Of course, templates may, under certain circumstances have fixed control information and not provide for user selections or parameter data entry.

- support plural, different control models regulating the use and/or auditing of either the same specific copy of electronic information content and/or differently regulating different copies (occurrences)

of the same electronic information content.

Differing models for billing, auditing, and security can be applied to the same piece of electronic information content and such differing sets of control information may employ, for control purposes, the same, or differing, granularities of electronic information control increments. This includes supporting variable control information for budgeting and auditing usage as applied to a variety of predefined increments of electronic information, including employing a variety of different budgets and/or metering increments for a given electronic information deliverable for: billing units of measure, credit limit, security budget limit and security content metering increments, and/or market surveying and customer profiling content metering increments. For example, a CD-ROM disk with a database of scientific articles might be in part billed according to a formula based on the number of bytes decrypted, number of articles containing said bytes decrypted, while a security budget might limit the use of said database to no more than 5% of the database per month for users on the wide area network it is installed on.

- provide mechanisms to persistently maintain trusted content usage and reporting control information through both a sufficiently secure chain of handling of content and content control information and through various forms of usage of such content wherein said persistence of control may survive such use. Persistence of control includes the ability to extract information from a VDE container object by creating a new container whose contents are at least in part secured and that contains both the extracted content and at least a portion of the control information which control information of the original container and/or are at least in part produced by control information of the original container for this purpose and/or VDE installation control information stipulates should persist and/or control usage of content in the newly formed container. Such control information can continue to manage usage of container content if the container is "embedded" into another VDE managed object, such as an object which contains plural embedded VDE containers, each of which contains content derived (extracted) from a different source.

- enables users, other value chain participants (such as clearinghouses and government agencies), and/or user organizations, to specify preferences or requirements related to their use of electronic content and/or appliances. Content users, such as end-user customers using commercially distributed content (games, information resources, software programs, etc.), can define, if allowed by senior control information, budgets, and/or other control information, to manage their own internal use of content. Uses include, for example, a user setting a limit on the price for electronic documents that the user is willing to pay without prior express user authorization, and the user establishing the character of metering information he or she is willing to allow to be collected (privacy protection). This includes providing the means for content users to protect the privacy of information derived from their use of a VDE installation and content and/or appliance usage auditing. In particular, VDE can prevent information related to a participant's usage of electronic content from being provided to other parties without the participant's tacit or explicit agreement.

- provide mechanisms that allow control information to "evolve" and be modified according, at least in part, to independently, securely delivered further control information. Said control information may include executable code (e.g., load modules) that has been certified as acceptable (e.g., reliable and trusted) for use with a specific VDE application, class of applications, and/or a VDE distributed arrangement. This modification (evolution) of control information can occur upon content control information (load modules and any associated data) circulating to one or more VDE participants in a pathway of handling of control information, or it may occur upon control information being received from a VDE participant. Handlers in a pathway of handling of content control information, to the extent each is authorized, can establish, modify, and/or contribute to, permission, auditing, payment, and reporting control information related to controlling, analyzing, paying for, and/or reporting usage of, electronic content and/or appliances (for example, as related to usage of VDE controlled property content). Independently delivered (from an independent source which is independent except in

regards to certification), at least in part secure, control information can be employed to securely modify content control information when content control information has flowed from one party to another party in a sequence of VDE content control information handling. This modification employs, for example, one or more VDE component assemblies being securely processed in a VDE secure subsystem. In an alternate embodiment, control information may be modified by a senior party through use of their VDE installation secure sub-system after receiving submitted, at least in part secured, control information from a "junior" party, normally in the form of a VDE administrative object. Control information passing along VDE pathways can represent a mixed control set, in that it may include: control information that persisted through a sequence of control information handlers, other control information that was allowed to be modified, and further control information representing new control information and/or mediating data. Such a control set represents an evolution of control information for disseminated content. In this example the overall content control

set for a VDE content container is "evolving" as it securely (e.g. communicated in encrypted form and using authentication and digital signaturing techniques) passes, at least in part, to a new participant's VDE installation where the proposed control information is securely received and handled. The received control information may be integrated (through use of the receiving parties' VDE installation secure sub-system) with in-place control information through a negotiation process involving both control information sets. For example, the modification, within the secure sub-system of a content provider's VDE installation, of content control information for a certain VDE content container may have occurred as a result of the incorporation of required control information provided by a financial credit provider. Said credit provider may have employed their VDE installation to prepare and securely communicate (directly or indirectly) said required control information to said content provider. Incorporating said required control information enables a content provider to allow the credit provider's credit to be employed by a content end-user to compensate for the end-user's

5 use of VDE controlled content and/or appliances, so long as said end-user has a credit account with said financial credit provider and said credit account has sufficient credit available. Similarly, control
10 information requiring the payment of taxes and/or the provision of revenue information resulting from electronic commerce activities may be securely received by a content provider. This control
15 information may be received, for example, from a government agency. Content providers might be required by law to incorporate such control information into the control information for commercially distributed content and/or services related to appliance usage. Proposed control
20 information is used to an extent allowed by senior control information and as determined by any negotiation trade-offs that satisfy priorities stipulated by each set (the received set and the proposed set). VDE also accommodates different control schemes specifically applying to different participants (e.g., individual participants and/or participant classes (types)) in a network of VDE content handling participants.

- support multiple simultaneous control models for the same content property and/or property portion. This allows, for example, for concurrent business activities which are dependent on electronic commercial product content distribution, such as acquiring detailed market survey information and/or supporting advertising, both of which can increase revenue and result in lower content costs to users and greater value to content providers. Such control information and/or overall control models may be applied, as determined or allowed by control information, in differing manners to different participants in a pathway of content, reporting, payment, and/or related control information handling. VDE supports applying different content control information to the same and/or different content and/or appliance usage related activities, and/or to different parties in a content and/or appliance usage model, such that different parties (or classes of VDE users, for example) are subject to differing control information managing their use of electronic information content. For example, differing control models based on the category of a user as a distributor of a VDE controlled content

object or an end-user of such content may result in different budgets being applied. Alternatively, for example, a one distributor may have the right to distribute a different array of properties than
5 another distributor (from a common content collection provided, for example, on optical disc). An individual, and/or a class or other grouping of end-users, may have different costs (for example, a student, senior citizen, and/or poor citizen user of
10 content who may be provided with the same or differing discounts) than a "typical" content user.

- support provider revenue information resulting from customer use of content and/or appliances, and/or
15 provider and/or end-user payment of taxes, through the transfer of credit and/or electronic currency from said end-user and/or provider to a government agency, might occur "automatically" as a result of such received control information causing the
20 generation of a VDE content container whose content includes customer content usage information reflecting secure, trusted revenue summary information and/or detailed user transaction listings (level of detail might depend, for example on type or

5

10

size of transaction—information regarding a bank interest payment to a customer or a transfer of a large (e.g. over \$10,000) might be, by law, automatically reported to the government). Such summary and/or detailed information related to taxable events and/or currency, and/or creditor currency transfer, may be passed along a pathway of reporting and/or payment to the government in a VDE container. Such a container may also be used for other VDE related content usage reporting information.

15

20

- support the flowing of content control information through different “branches” of content control information handling so as to accommodate, under the present invention’s preferred embodiment, diverse controlled distributions of VDE controlled content. This allows different parties to employ the same initial electronic content with differing (perhaps competitive) control strategies. In this instance, a party who first placed control information on content can make certain control assumptions and these assumptions would evolve into more specific and/or extensive control

assumptions. These control assumptions can evolve during the branching sequence upon content model participants submitting control information changes, for example, for use in "negotiating" with "in place" content control information. This can result in new or modified content control information and/or it might involve the selection of certain one or more already "in-place" content usage control methods over in-place alternative methods, as well as the submission of relevant control information parameter data. This form of evolution of different control information sets applied to different copies of the same electronic property content and/or appliance results from VDE control information flowing "down" through different branches in an overall pathway of handling and control and being modified differently as it diverges down these different pathway branches. This ability of the present invention to support multiple pathway branches for the flow of both VDE content control information and VDE managed content enables an electronic commerce marketplace which supports diverging, competitive business partnerships, agreements, and evolving overall business models

which can employ the same content properties combined, for example, in differing collections of content representing differing at least in part competitive products.

5

- enable a user to securely extract, through the use of the secure subsystem at the user's VDE installation, at least a portion of the content included within a VDE content container to produce a new, secure object (content container), such that the extracted information is maintained in a continually secure manner through the extraction process. Formation of the new VDE container containing such extracted content shall result in control information consistent with, or specified by, the source VDE content container, and/or local VDE installation secure subsystem as appropriate, content control information. Relevant control information, such as security and administrative information, derived, at least in part, from the parent (source) object's control information, will normally be automatically inserted into a new VDE content container object containing extracted VDE content. This process typically occurs under the control framework of a

10

15

20

parent object and/or VDE installation control
information executing at the user's VDE installation
secure subsystem (with, for example, at least a
portion of this inserted control information being
5 stored securely in encrypted form in one or more
permissions records). In an alternative embodiment,
the derived content control information applied to
extracted content may be in part or whole derived
from, or employ, content control information stored
10 remotely from the VDE installation that performed
the secure extraction such as at a remote server
location. As with the content control information for
most VDE managed content, features of the present
invention allows the content's control information to:

15

- (a) "evolve," for example, the extractor of content
may add new control methods and/or modify
control parameter data, such as VDE
application compliant methods, to the extent
20 allowed by the content's in-place control
information. Such new control information
might specify, for example, who may use at
least a portion of the new object, and/or how
said at least a portion of said extracted

content may be used (e.g. when at least a portion may be used, or what portion or quantity of portions may be used);

5

- (b) allow a user to combine additional content with at least a portion of said extracted content, such as material authored by the extractor and/or content (for example, images, video, audio, and/or text) extracted from one or more other VDE container objects for placement directly into the new container;

10

15

- (c) allow a user to securely edit at least a portion of said content while maintaining said content in a secure form within said VDE content container;

20

- (d) append extracted content to a pre-existing VDE content container object and attach associated control information -- in these cases, user added information may be secured, e.g., encrypted, in part or as a whole, and may be subject to usage and/or auditing control

information that differs from the those applied
to previously in place object content;

- 5 (e) preserve VDE control over one or more
portions of extracted content after various
forms of usage of said portions, for example,
maintain content in securely stored form
while allowing "temporary" on screen display
of content or allowing a software program to
10 be maintained in secure form but transiently
decrypt any encrypted executing portion of
said program (all, or only a portion, of said
program may be encrypted to secure the
program).

15
Generally, the extraction features of the present
invention allow users to aggregate and/or
disseminate and/or otherwise use protected
electronic content information extracted from
20 content container sources while maintaining secure
VDE capabilities thus preserving the rights of
providers in said content information after various
content usage processes.

• support the aggregation of portions of VDE controlled content, such portions being subject to differing VDE content container control information, wherein various of said portions may have been provided by independent, different content providers from one or more different locations remote to the user performing the aggregation. Such aggregation, in the preferred embodiment of the present invention, may involve preserving at least a portion of the control information (e.g., executable code such as load modules) for each of various of said portions by, for example, embedding some or all of such portions individually as VDE content container objects within an overall VDE content container and/or embedding some or all of such portions directly into a VDE content container. In the latter case, content control information of said content container may apply differing control information sets to various of such portions based upon said portions original control information requirements before aggregation. Each of such embedded VDE content containers may have its own control information in the form of one or more permissions records. Alternatively, a negotiation between

control information associated with various aggregated portions of electronic content, may produce a control information set that would govern some or all of the aggregated content portions. The VDE content control information produced by the negotiation may be uniform (such as having the same load modules and/or component assemblies, and/or it may apply differing such content control information to two or more portions that constitute an aggregation of VDE controlled content such as differing metering, budgeting, billing and/or payment models. For example, content usage payment may be automatically made, either through a clearinghouse, or directly, to different content providers for different portions.

- enable flexible metering of, or other collection of information related to, use of electronic content and/or electronic appliances. A feature of the present invention enables such flexibility of metering control mechanisms to accommodate a simultaneous, broad array of: (a) different parameters related to electronic information content use; (b) different increment units (bytes, documents,

properties, paragraphs, images, etc.) and/or other organizations of such electronic content; and/or (c) different categories of user and/or VDE installation types, such as client organizations, departments, projects, networks, and/or individual users, etc.

This feature of the present invention can be employed for content security, usage analysis (for example, market surveying), and/or compensation based upon the use and/or exposure to VDE managed content. Such metering is a flexible basis for ensuring payment for content royalties, licensing, purchasing, and/or advertising. A feature of the present invention provides for payment means supporting flexible electronic currency and credit mechanisms, including the ability to securely maintain audit trails reflecting information related to use of such currency or credit. VDE supports multiple differing hierarchies of client organization control information wherein an organization client administrator distributes control information specifying the usage rights of departments, users, and/or projects. Likewise, a department (division) network manager can function as a distributor

(budgets, access rights, etc.) for department networks, projects, and/or users, etc.

- 5 ● provide scalable, integratable, standardized control means for use on electronic appliances ranging from inexpensive consumer (for example, television set-top appliances) and professional devices (and hand-held PDAs) to servers, mainframes, communication switches, etc. The scalable
10 transaction management/auditing technology of the present invention will result in more efficient and reliable interoperability amongst devices functioning in electronic commerce and/or data security environments. As standardized physical containers
15 have become essential to the shipping of physical goods around the world, allowing these physical containers to universally "fit" unloading equipment, efficiently use truck and train space, and accommodate known arrays of objects (for example,
20 boxes) in an efficient manner, so VDE electronic content containers may, as provided by the present invention, be able to efficiently move electronic information content (such as commercially published properties, electronic currency and credit, and

content audit information), and associated content control information, around the world.

Interoperability is fundamental to efficient

electronic commerce. The design of the VDE

5 foundation, VDE load modules, and VDE containers,

are important features that enable the VDE node

operating environment to be compatible with a very

broad range of electronic appliances. The ability, for

example, for control methods based on load modules

10 to execute in very "small" and inexpensive secure

sub-system environments, such as environments

with very little read/write memory, while also being

able to execute in large memory sub-systems that

may be used in more expensive electronic

15 appliances, supports consistency across many

machines. This consistent VDE operating

environment, including its control structures and

container architecture, enables the use of

standardized VDE content containers across a broad

20 range of device types and host operating

environments. Since VDE capabilities can be

seamlessly integrated as extensions, additions,

and/or modifications to fundamental capabilities of

electronic appliances and host operating systems,

VDE containers, content control information, and the VDE foundation will be able to work with many device types and these device types will be able to consistently and efficiently interpret and enforce VDE control information. Through this integration users can also benefit from a transparent interaction with many of the capabilities of VDE. VDE integration with software operating on a host electronic appliance supports a variety of capabilities that would be unavailable or less secure without such integration. Through integration with one or more device applications and/or device operating environments, many capabilities of the present invention can be presented as inherent capabilities of a given electronic appliance, operating system, or appliance application. For example, features of the present invention include:

(a) VDE system software to in part extend and/or modify host operating systems such that they possesses VDE capabilities, such as enabling secure transaction processing and electronic information storage; (b) one or more application programs that in part represent tools associated with VDE operation; and/or (c) code to be integrated into application

5 programs, wherein such code incorporates references
into VDE system software to integrate VDE
capabilities and makes such applications VDE
aware (for example, word processors, database
10 retrieval applications, spreadsheets, multimedia
presentation authoring tools, film editing software,
music editing software such as MIDI applications
and the like, robotics control systems such as those
associated with CAD/CAM environments and NCM
15 software and the like, electronic mail systems,
teleconferencing software, and other data authoring,
creating, handling, and/or usage applications
including combinations of the above). These one or
more features (which may also be implemented in
20 firmware or hardware) may be employed in
conjunction with a VDE node secure hardware
processing capability, such as a microcontroller(s),
microprocessor(s), other CPU(s) or other digital
processing logic.

20

- employ audit reconciliation and usage pattern
evaluation processes that assess, through certain,
normally network based, transaction processing
reconciliation and threshold checking activities,

whether certain violations of security of a VDE arrangement have occurred. These processes are performed remote to VDE controlled content end-user VDE locations by assessing, for example, purchases, and/or requests, for electronic properties by a given VDE installation. Applications for such reconciliation activities include assessing whether the quantity of remotely delivered VDE controlled content corresponds to the amount of financial credit and/or electronic currency employed for the use of such content. A trusted organization can acquire information from content providers concerning the cost for content provided to a given VDE installation and/or user and compare this cost for content with the credit and/or electronic currency disbursements for that installation and/or user. Inconsistencies in the amount of content delivered versus the amount of disbursement can prove, and/or indicate, depending on the circumstances, whether the local VDE installation has been, at least to some degree, compromised (for example, certain important system security functions, such as breaking encryption for at least some portion of the secure subsystem and/or VDE controlled content by uncovering one or more

5

10

15

20

keys). Determining whether irregular patterns (e.g. unusually high demand) of content usage, or requests for delivery of certain kinds of VDE controlled information during a certain time period by one or more VDE installations and/or users (including, for example, groups of related users whose aggregate pattern of usage is suspicious) may also be useful in determining whether security at such one or more installations, and/or by such one or more users, has been compromised, particularly when used in combination with an assessment of electronic credit and/or currency provided to one or more VDE users and/or installations, by some or all of their credit and/or currency suppliers, compared with the disbursements made by such users and/or installations.

- support security techniques that materially increase the time required to "break" a system's integrity. This includes using a collection of techniques that minimizes the damage resulting from comprising some aspect of the security features of the present inventions.

- provide a family of authoring, administrative, reporting, payment, and billing tool user applications that comprise components of the present invention's trusted/secure, universe wide, distributed transaction control and administration system. These components support VDE related: object creation (including placing control information on content), secure object distribution and management (including distribution control information, financial related, and other usage analysis), client internal VDE activities administration and control, security management, user interfaces, payment disbursement, and clearinghouse related functions. These components are designed to support highly secure, uniform, consistent, and standardized: electronic commerce and/or data security pathway(s) of handling, reporting, and/or payment; content control and administration; and human factors (e.g. user interfaces).
- support the operation of a plurality of clearinghouses, including, for example, both financial and user clearinghouse activities, such as

those performed by a client administrator in a large organization to assist in the organization's use of a VDE arrangement, including usage information analysis, and control of VDE activities by

5 individuals and groups of employees such as specifying budgets and the character of usage rights available under VDE for certain groups of and/or individual, client personnel, subject to control

10 information series to control information submitted by the client administrator. At a clearinghouse, one or more VDE installations may operate together with a trusted distributed database environment (which may include concurrent database processing means). A financial clearinghouse normally receives

15 at its location securely delivered content usage information, and user requests (such as requests for further credit, electronic currency, and/or higher credit limit). Reporting of usage information and user requests can be used for supporting electronic

20 currency, billing, payment and credit related activities, and/or for user profile analysis and/or broader market survey analysis and marketing (consolidated) list generation or other information derived, at least in part, from said usage

information. this information can be provided to content providers or other parties, through secure, authenticated encrypted communication to the VDE installation secure subsystems. Clearinghouse processing means would normally be connected to specialized I/O means, which may include high speed telecommunication switching means that may be used for secure communications between a clearinghouse and other VDE pathway participants.

- securely support electronic currency and credit usage control, storage, and communication at, and between, VDE installations. VDE further supports automated passing of electronic currency and/or credit information, including payment tokens (such as in the form of electronic currency or credit) or other payment information, through a pathway of payment, which said pathway may or may not be the same as a pathway for content usage information reporting. Such payment may be placed into a VDE container created automatically by a VDE installation in response to control information stipulating the "withdrawal" of credit or electronic currency from an electronic credit or currency

5 account based upon an amount owed resulting from
usage of VDE controlled electronic content and/or
appliances. Payment credit or currency may then be
automatically communicated in protected (at least in
10 part encrypted) form through telecommunication of
a VDE container to an appropriate party such as a
clearinghouse, provider of original property content
or appliance, or an agent for such provider (other
than a clearinghouse). Payment information may be
packaged in said VDE content container with, or
without, related content usage information, such as
metering information. An aspect of the present
invention further enables certain information
regarding currency use to be specified as
15 unavailable to certain, some, or all VDE parties
("conditionally" to fully anonymous currency) and/or
further can regulate certain content information,
such as currency and/or credit use related
information (and/or other electronic information
20 usage data) to be available only under certain strict
circumstances, such as a court order (which may
itself require authorization through the use of a
court controlled VDE installation that may be
required to securely access "conditionally"

anonymous information). Currency and credit information, under the preferred embodiment of the present invention, is treated as administrative content;

5

- support fingerprinting (also known as watermarking) for embedding in content such that when content protected under the present invention is released in clear form from a VDE object (displayed, printed, communicated, extracted, and/or saved), information representing the identification of the user and/or VDE installation responsible for transforming the content into clear form is embedded into the released content. Fingerprinting is useful in providing an ability to identify who extracted information in clear form a VDE container, or who made a copy of a VDE object or a portion of its contents. Since the identity of the user and/or other identifying information may be embedded in an obscure or generally concealed manner, in VDE container content and/or control information, potential copyright violators may be deterred from unauthorized extraction or copying. Fingerprinting normally is embedded into

10

15

20

unencrypted electronic content or control information, though it can be embedded into encrypted content and later placed in unencrypted content in a secure VDE installation sub-system as

5 the encrypted content carrying the fingerprinting information is decrypted. Electronic information, such as the content of a VDE container, may be fingerprinted as it leaves a network (such as Internet) location bound for a receiving party. Such

10 repository information may be maintained in unencrypted form prior to communication and be encrypted as it leaves the repository. Fingerprinting would preferably take place as the content leaves the repository, but before the encryption step.

15 Encrypted repository content can be decrypted, for example in a secure VDE sub-system, fingerprint information can be inserted, and then the content can be re-encrypted for transmission. Embedding

20 identification information of the intended recipient user and/or VDE installation into content as it leaves, for example, an Internet repository, would provide important information that would identify or assist in identifying any party that managed to compromise the security of a VDE installation or the

5 delivered content. If a party produces an authorized clear form copy of VDE controlled content, including making unauthorized copies of an authorized clear form copy, fingerprint information would point back to that individual and/or his or her VDE installation. Such hidden information will act as a strong disincentive that should dissuade a substantial portion of potential content "pirates" from stealing other parties electronic information.

10 Fingerprint information identifying a receiving party and/or VDE installation can be embedded into a VDE object before, or during, decryption, replication, or communication of VDE content objects to receivers. Fingerprinting electronic content before it is encrypted for transfer to a

15 customer or other user provides information that can be very useful for identifying who received certain content which may have then been distributed or made available in unencrypted form.

20 This information would be useful in tracking who may have "broken" the security of a VDE installation and was illegally making certain electronic content available to others. Fingerprinting may provide additional, available

information such as time and/or date of the release
(for example extraction) of said content information.
Locations for inserting fingerprints may be specified
by VDE installation and/or content container control
5 information. This information may specify that
certain areas and/or precise locations within
properties should be used for fingerprinting, such as
one or more certain fields of information or
information types. Fingerprinting information may
10 be incorporated into a property by modifying in a
normally undetectable way color frequency and/or
the brightness of certain image pixels, by slightly
modifying certain audio signals as to frequency, by
modifying font character formation, etc.
15 Fingerprint information, itself, should be encrypted
so as to make it particularly difficult for tampered
fingerprints to be interpreted as valid. Variations in
fingerprint locations for different copies of the same
property; "false" fingerprint information; and
20 multiple copies of fingerprint information within a
specific property or other content which copies
employ different fingerprinting techniques such as
information distribution patterns, frequency and/or
brightness manipulation, and encryption related

techniques, are features of the present invention for increasing the difficulty of an unauthorized individual identifying fingerprint locations and erasing and/or modifying fingerprint information.

5

10

15

20

- provide smart object agents that can carry requests, data, and/or methods, including budgets, authorizations, credit or currency, and content. For example, smart objects may travel to and/or from remote information resource locations and fulfill requests for electronic information content. Smart objects can, for example, be transmitted to a remote location to perform a specified database search on behalf of a user or otherwise "intelligently" search remote one or more repositories of information for user desired information. After identifying desired information at one or more remote locations, by for example, performing one or more database searches, a smart object may return via communication to the user in the form of a secure "return object" containing retrieved information. A user may be charged for the remote retrieving of information, the returning of information to the user's VDE installation, and/or the use of such information. In

- the latter case, a user may be charged only for the information in the return object that the user actually uses. Smart objects may have the means to request use of one or more services and/or resources.
- 5 Services include locating other services and/or resources such as information resources, language or format translation, processing, credit (or additional credit) authorization, etc. Resources include reference databases, networks, high powered or
- 10 specialized computing resources (the smart object may carry information to another computer to be efficiently processed and then return the information to the sending VDE installation), remote object repositories, etc. Smart objects can
- 15 make efficient use of remote resources (e.g. centralized databases, super computers, etc.) while providing a secure means for charging users based on information and/or resources actually used.
- 20
- support both "translations" of VDE electronic agreements elements into modern language printed agreement elements (such as English language agreements) and translations of electronic rights protection/transaction management modern

language agreement elements to electronic VDE
agreement elements. This feature requires
maintaining a library of textual language that
corresponds to VDE load modules and/or methods
5 and/or component assemblies. As VDE methods are
proposed and/or employed for VDE agreements, a
listing of textual terms and conditions can be
produced by a VDE user application which, in a
preferred embodiment, provides phrases, sentences
10 and/or paragraphs that have been stored and
correspond to said methods and/or assemblies. This
feature preferably employs artificial intelligence
capabilities to analyze and automatically determine,
and/or assist one or more users to determine, the
15 proper order and relationship between the library
elements corresponding to the chosen methods
and/or assemblies so as to compose some or all
portions of a legal or descriptive document. One or
more users, and/or preferably an attorney (if the
20 document a legal, binding agreement), would review
the generated document material upon completion
and employ such additional textual information
and/or editing as necessary to describe non
electronic transaction elements of the agreement

and make any other improvements that may be necessary. These features further support employing modern language tools that allow one or more users to make selections from choices and provide answers to questions and to produce a VDE electronic agreement from such a process. This process can be interactive and the VDE agreement formulation process may employ artificial intelligence expert system technology that learns from responses and, where appropriate and based at least in part on said responses, provides further choices and/or questions which "evolves" the desired VDE electronic agreement.

- support the use of multiple VDE secure subsystems in a single VDE installation. Various security and/or performance advantages may be realized by employing a distributed VDE design within a single VDE installation. For example, designing a hardware based VDE secure subsystem into an electronic appliance VDE display device, and designing said subsystem's integration with said display device so that it is as close as possible to the point of display, will increase the security for video

5 materials by making it materially more difficult to
"steal" decrypted video information as it moves from
outside to inside the video system. Ideally, for
example, a VDE secure hardware module would be
in the same physical package as the actual display
monitor, such as within the packaging of a video
monitor or other display device, and such device
would be designed, to the extent commercially
practical, to be as tamper resistant as reasonable.
10 As another example, embedding a VDE hardware
module into an I/O peripheral may have certain
advantages from the standpoint of overall system
throughput. If multiple VDE instances are
employed within the same VDE installation, these
15 instances will ideally share resources to the extent
practical, such as VDE instances storing certain
control information and content and/or appliance
usage information on the same mass storage device
and in the same VDE management database.
20

- requiring reporting and payment compliance by
employing exhaustion of budgets and time ageing of
keys. For example, a VDE commercial arrangement
and associated content control information may

involve a content provider's content and the use of clearinghouse credit for payment for end-user usage of said content. Control information regarding said arrangement may be delivered to a user's (of said
5 content) VDE installation and/or said financial clearinghouse's VDE installation. Said control information might require said clearinghouse to prepare and telecommunicate to said content provider both content usage based information in a
10 certain form, and content usage payment in the form of electronic credit (such credit might be "owned" by the provider after receipt and used in lieu of the availability or adequacy of electronic currency) and/or electronic currency. This delivery of
15 information and payment may employ trusted VDE installation secure subsystems to securely, and in some embodiments, automatically, provide in the manner specified by said control information, said usage information and payment content. Features
20 of the present invention help ensure that a requirement that a clearinghouse report such usage information and payment content will be observed. For example, if one participant to a VDE electronic agreement fails to observe such information

reporting and/or paying obligation, another participant can stop the delinquent party from successfully participating in VDE activities related to such agreement. For example, if required usage information and payment was not reported as specified by content control information, the "injured" party can fail to provide, through failing to securely communicate from his VDE installation secure subsystem, one or more pieces of secure information necessary for the continuance of one or more critical processes. For example, failure to report information and/or payment from a clearinghouse to a content provider (as well as any security failures or other disturbing irregularities) can result in the content provider not providing key and/or budget refresh information to the clearinghouse, which information can be necessary to authorize use of the clearinghouse's credit for usage of the provider's content and which the clearinghouse would communicate to end-user's during a content usage reporting communication between the clearinghouse and end-user. As another example, a distributor that failed to make payments and/or report usage information to a

5

10

content provider might find that their budget for creating permissions records to distribute the content provider's content to users, and/or a security budget limiting one or more other aspect of their use of the provider's content, are not being refreshed by the content provider, once exhausted or timed-out (for example, at a predetermined date). In these and other cases, the offended party might decide not to refresh time ageing keys that had "aged out." Such a use of time aged keys has a similar impact as failing to refresh budgets or time-aged authorizations.

15

20

- support smart card implementations of the present invention in the form of portable electronic appliances, including cards that can be employed as secure credit, banking, and/or money cards. A feature of the present invention is the use of portable VDEs as transaction cards at retail and other establishments, wherein such cards can "dock" with an establishment terminal that has a VDE secure sub-system and/or an online connection to a VDE secure and/or otherwise secure and compatible subsystem, such as a "trusted" financial

clearinghouse (e.g., VISA, Mastercard). The VDE card and the terminal (and/or online connection) can securely exchange information related to a transaction, with credit and/or electronic currency being transferred to a merchant and/or clearinghouse and transaction information flowing back to the card. Such a card can be used for transaction activities of all sorts. A docking station, such as a PCMCIA connector on an electronic appliance, such as a personal computer, can receive a consumer's VDE card at home. Such a station/card combination can be used for on-line transactions in the same manner as a VDE installation that is permanently installed in such an electronic appliance. The card can be used as an "electronic wallet" and contain electronic currency as well as credit provided by a clearinghouse. The card can act as a convergence point for financial activities of a consumer regarding many, if not all, merchant, banking, and on-line financial transactions, including supporting home banking activities. A consumer can receive his paycheck and/or investment earnings and/or "authentic" VDE content container secured detailed information on such

receipts, through on-line connections. A user can send digital currency to another party with a VDE arrangement, including giving away such currency. A VDE card can retain details of transactions in a highly secure and database organized fashion so that financially related information is both consolidated and very easily retrieved and/or analyzed. Because of the VDE security, including use of effective encryption, authentication, digital signaturing, and secure database structures, the records contained within a VDE card arrangement may be accepted as valid transaction records for government and/or corporate recordkeeping requirements. In some embodiments of the present invention a VDE card may employ docking station and/or electronic appliance storage means and/or share other VDE arrangement means local to said appliance and/or available across a network, to augment the information storage capacity of the VDE card, by for example, storing dated, and/or archived, backup information. Taxes relating to some or all of an individual's financial activities may be automatically computed based on "authentic" information securely stored and available to said

VDE card. Said information may be stored in said card, in said docking station, in an associated electronic appliance, and/or other device operatively attached thereto, and/or remotely, such as at a remote server site. A card's data, e.g. transaction history, can be backed up to an individual's personal computer or other electronic appliance and such an appliance may have an integrated VDE installation of its own. A current transaction, recent transactions (for redundancy), or all or other selected card data may be backed up to a remote backup repository, such a VDE compatible repository at a financial clearinghouse, during each or periodic docking for a financial transaction and/or information communication such as a user/merchant transaction. Backing up at least the current transaction during a connection with another party's VDE installation (for example a VDE installation that is also on a financial or general purpose electronic network), by posting transaction information to a remote clearinghouse and/or bank, can ensure that sufficient backup is conducted to enable complete reconstruction of VDE card internal information in the event of a card failure or loss.

- support certification processes that ensure authorized interoperability between various VDE installations so as to prevent VDE arrangements and/or installations that unacceptably deviate in specification protocols from other VDE arrangements and/or installations from interoperating in a manner that may introduce security (integrity and/or confidentiality of VDE secured information), process control, and/or software compatibility problems. Certification validates the identity of VDE installations and/or their components, as well as VDE users. Certification data can also serve as information that contributes to determining the decommissioning or other change related to VDE sites.
- support the separation of fundamental transaction control processes through the use of event (triggered) based method control mechanisms. These event methods trigger one or more other VDE methods (which are available to a secure VDE sub-system) and are used to carry out VDE managed transaction related processing. These triggered methods include independently (separably) and

securely processable component billing management methods, budgeting management methods, metering management methods, and related auditing management processes. As a result of this feature of the present invention, independent triggering of metering, auditing, billing, and budgeting methods, the present invention is able to efficiently, concurrently support multiple financial currencies (e.g. dollars, marks, yen) and content related budgets, and/or billing increments as well as very flexible content distribution models.

- support, complete, modular separation of the control structures related to (1) content event triggering, (2) auditing, (3) budgeting (including specifying no right of use or unlimited right of use), (4) billing, and (5) user identity (VDE installation, client name, department, network, and/or user, etc.). The independence of these VDE control structures provides a flexible system which allows plural relationships between two or more of these structures, for example, the ability to associate a financial budget with different event trigger structures (that are put in place to enable

controlling content based on its logical portions).

Without such separation between these basic VDE capabilities, it would be more difficult to efficiently maintain separate metering, budgeting, identification, and/or billing activities which involve the same, differing (including overlapping), or entirely different, portions of content for metering, billing, budgeting, and user identification, for example, paying fees associated with usage of content, performing home banking, managing advertising services, etc. VDE modular separation of these basic capabilities supports the programming of plural, "arbitrary" relationships between one or differing content portions (and/or portion units) and budgeting, auditing, and/or billing control information. For example, under VDE, a budget limit of \$200 dollars or 300 German Marks a month may be enforced for decryption of a certain database and 2 U.S. Dollars or 3 German Marks may be charged for each record of said database decrypted (depending on user selected currency). Such usage can be metered while an additional audit for user profile purposes can be prepared recording the identity of each filed displayed. Additionally,

5 further metering can be conducted regarding the
number of said database bytes that have been
decrypted, and a related security budget may
prevent the decrypting of more than 5% of the total
bytes of said database per year. The user may also,
under VDE (if allowed by senior control
information), collect audit information reflecting
usage of database fields by different individuals and
client organization departments and ensure that
10 differing rights of access and differing budgets
limiting database usage can be applied to these
client individuals and groups. Enabling content
providers and users to practically employ such
diverse sets of user identification, metering,
15 budgeting, and billing control information results, in
part, from the use of such independent control
capabilities. As a result, VDE can support great
configurability in creation of plural control models
applied to the same electronic property and the
20 same and/or plural control models applied to
differing or entirely different content models (for
example, home banking versus electronic shopping).

Methods, Other Control Information, and VDE Objects

VDE control information (e.g., methods) that collectively control use of VDE managed properties (database, document, individual commercial product), are either shipped with the content itself (for example, in a content container) and/or one or more portions of such control information is shipped to distributors and/or other users in separably deliverable "administrative objects." A subset of the methods for a property may in part be delivered with each property while one or more other subsets of methods can be delivered separately to a user or otherwise made available for use (such as being available remotely by telecommunication means). Required methods (methods listed as required for property and/or appliance use) must be available as specified if VDE controlled content (such as intellectual property distributed within a VDE content container) is to be used. Methods that control content may apply to a plurality of VDE container objects, such as a class or other grouping of such objects. Methods may also be required by certain users or classes of users and/or VDE installations and/or classes of installations for such parties to use one or more specific, or classes of, objects.

A feature of VDE provided by the present invention is that certain one or more methods can be specified as required in order

for a VDE installation and/or user to be able to use certain
and/or all content. For example, a distributor of a certain type of
content might be allowed by "senior" participants (by content
creators, for example) to require a method which prohibits
5 end-users from electronically saving decrypted content, a
provider of credit for VDE transactions might require an audit
method that records the time of an electronic purchase, and/or a
user might require a method that summarizes usage information
for reporting to a clearinghouse (e.g. billing information) in a
10 way that does not convey confidential, personal information
regarding detailed usage behavior.

A further feature of VDE provided by the present
invention is that creators, distributors, and users of content can
15 select from among a set of predefined methods (if available) to
control container content usage and distribution functions and/or
they may have the right to provide new customized methods to
control at least certain usage functions (such "new" methods may
be required to be certified for trustedness and interoperability to
20 the VDE installation and/or for of a group of VDE applications).
As a result, VDE provides a very high degree of configurability
with respect to how the distribution and other usage of each
property or object (or one or more portions of objects or properties
as desired and/or applicable) will be controlled. Each VDE

participant in a VDE pathway of content control information
may set methods for some or all of the content in a VDE
container, so long as such control information does not conflict
with senior control information already in place with respect to:

5

- (1) certain or all VDE managed content,
- (2) certain one or more VDE users and/or groupings of
users,
- (3) certain one or more VDE nodes and/or groupings of
nodes, and/or
- (4) certain one or more VDE applications and/or
arrangements.

10

15

20

For example, a content creator's VDE control information
for certain content can take precedence over other submitted
VDE participant control information and, for example, if allowed
by senior control information, a content distributor's control
information may itself take precedence over a client
administrator's control information, which may take precedence
over an end-user's control information. A path of distribution
participant's ability to set such electronic content control

information can be limited to certain control information (for example, method mediating data such as pricing and/or sales dates) or it may be limited only to the extent that one or more of the participant's proposed control information conflicts with control information set by senior control information submitted previously by participants in a chain of handling of the property, or managed in said participant's VDE secure subsystem.

VDE control information may, in part or in full, (a) represent control information directly put in place by VDE content control information pathway participants, and/or (b) comprise control information put in place by such a participant on behalf of a party who does not directly handle electronic content (or electronic appliance) permissions records information (for example control information inserted by a participant on behalf of a financial clearinghouse or government agency). Such control information methods (and/or load modules and/or mediating data and/or component assemblies) may also be put in place by either an electronic automated, or a semi-automated and human assisted, control information (control set) negotiating process that assesses whether the use of one or more pieces of submitted control information will be integrated into and/or replace existing control information (and/or chooses between alternative control information based upon interaction with

in-place control information) and how such control information may be used.

Control information may be provided by a party who does
5 not directly participate in the handling of electronic content
(and/or appliance) and/or control information for such content
(and/or appliance). Such control information may be provided in
secure form using VDE installation secure sub-system managed
communications (including, for example, authenticating the
10 deliverer of at least in part encrypted control information)
between such not directly participating one or more parties' VDE
installation secure subsystems, and a pathway of VDE content
control information participant's VDE installation secure
subsystem. This control information may relate to, for example,
15 the right to access credit supplied by a financial services
provider, the enforcement of regulations or laws enacted by a
government agency, or the requirements of a customer of VDE
managed content usage information (reflecting usage of content
by one or more parties other than such customer) relating to the
20 creation, handling and/or manner of reporting of usage
information received by such customer. Such control information
may, for example, enforce societal requirements such as laws
related to electronic commerce.

VDE content control information may apply differently to different pathway of content and/or control information handling participants. Furthermore, permissions records rights may be added, altered, and/or removed by a VDE participant if they are
5 allowed to take such action. Rights of VDE participants may be defined in relation to specific parties and/or categories of parties and/or other groups of parties in a chain of handling of content and/or content control information (e.g., permissions records). Modifications to control information that may be made by a
10 given, eligible party or parties, may be limited in the number of modifications, and/or degree of modification, they may make.

At least one secure subsystem in electronic appliances of creators, distributors, auditors, clearinghouses, client
15 administrators, and end-users (understanding that two or more of the above classifications may describe a single user) provides a "sufficiently" secure (for the intended applications) environment for:

- 20
1. Decrypting properties and control information;
 2. Storing control and metering related information;
 3. Managing communications;

4. Processing core control programs, along with associated data, that constitute control information for electronic content and/or appliance rights protection, including the enforcing of preferences and requirements of VDE participants.

Normally, most usage, audit, reporting, payment, and distribution control methods are themselves at least in part encrypted and are executed by the secure subsystem of a VDE installation. Thus, for example, billing and metering records can be securely generated and updated, and encryption and decryption keys are securely utilized, within a secure subsystem. Since VDE also employs secure (e.g. encrypted and authenticated) communications when passing information between the participant location (nodes) secure subsystems of a VDE arrangement, important components of a VDE electronic agreement can be reliably enforced with sufficient security (sufficiently trusted) for the intended commercial purposes. A VDE electronic agreement for a value chain can be composed, at least in part, of one or more subagreements between one or more subsets of the value chain participants. These subagreements are comprised of one or more electronic contract "compliance" elements (methods including associated parameter data) that ensure the protection of the rights of VDE participants.

5 The degree of trustedness of a VDE arrangement will be primarily based on whether hardware SPUs are employed at participant location secure subsystems and the effectiveness of the SPU hardware security architecture, software security techniques when an SPU is emulated in software, and the encryption algorithm(s) and keys that are employed for securing content, control information, communications, and access to VDE node (VDE installation) secure subsystems. Physical facility and user identity authentication security procedures may be used instead of hardware SPUs at certain nodes, such as at an established financial clearinghouse, where such procedures may provide sufficient security for trusted interoperability with a VDE arrangement employing hardware SPUs at user nodes.

15 The updating of property management files at each location of a VDE arrangement, to accommodate new or modified control information, is performed in the VDE secure subsystem and under the control of secure management file updating programs executed by the protected subsystem. Since all secure communications are at least in part encrypted and the processing inside the secure subsystem is concealed from outside observation and interference, the present invention ensures that content control information can be enforced. As a result, the creator and/or distributor and/or client administrator and/or

other contributor of secure control information for each property
(for example, an end-user restricting the kind of audit
information he or she will allow to be reported and/or a financial
clearinghouse establishing certain criteria for use of its credit for
5 payment for use of distributed content) can be confident that
their contributed and accepted control information will be
enforced (within the security limitations of a given VDE security
implementation design). This control information can determine,
for example:

10

(1) How and/or to whom electronic content can be
provided, for example, how an electronic property
can be distributed;

15

(2) How one or more objects and/or properties, or
portions of an object or property, can be directly
used, such as decrypted, displayed, printed, etc;

20

(3) How payment for usage of such content and/or
content portions may or must be handled; and

(4) How audit information about usage information
related to at least a portion of a property should be
collected, reported, and/or used.

Seniority of contributed control information, including resolution of conflicts between content control information submitted by multiple parties, is normally established by:

- 5 (1) the sequence in which control information is put in place by various parties (in place control information normally takes precedence over subsequently submitted control information),
- 10 (2) the specifics of VDE content and/or appliance control information. For example, in-place control information can stipulate which subsequent one or more piece of control from one or more parties or class of parties will take precedence over control
15 information submitted by one or more yet different parties and/or classes of parties, and/or
- 20 (3) negotiation between control information sets from plural parties, which negotiation establishes what control information shall constitute the resulting control information set for a given piece of VDE managed content and/or VDE installation.

Electronic Agreements and Rights Protection

An important feature of VDE is that it can be used to assure the administration of, and adequacy of security and rights protection for, electronic agreements implemented through the use of the present invention. Such agreements may involve one or more of:

- (1) creators, publishers, and other distributors, of electronic information,
- (2) financial service (e.g. credit) providers,
- (3) users of (other than financial service providers) information arising from content usage such as content specific demographic information and user specific descriptive information. Such users may include market analysts, marketing list compilers for direct and directed marketing, and government agencies,
- (4) end users of content,
- (5) infrastructure service and device providers such as telecommunication companies and hardware

manufacturers (semiconductor and electronic
appliance and/or other computer system
manufacturers) who receive compensation based
upon the use of their services and/or devices, and

5

- (6) certain parties described by electronic information.

VDE supports commercially secure "extended" value chain
electronic agreements. VDE can be configured to support the
various underlying agreements between parties that comprise
this extended agreement. These agreements can define
important electronic commerce considerations including:

10

- (1) security,
- (2) content use control, including electronic distribution,
- (3) privacy (regarding, for example, information
concerning parties described by medical, credit, tax,
personal, and/or of other forms of confidential
information),
- (4) management of financial processes, and

15

20

- (5) pathways of handling for electronic content, content and/or appliance control information, electronic content and/or appliance usage information and payment and/or credit.

5

VDE agreements may define the electronic commerce relationship of two or more parties of a value chain, but such agreements may, at times, not directly obligate or otherwise directly involve other VDE value chain participants. For example, an electronic agreement between a content creator and a distributor may establish both the price to the distributor for a creator's content (such as for a property distributed in a VDE container object) and the number of copies of this object that this distributor may distribute to end-users over a given period of time. In a second agreement, a value chain end-user may be involved in a three party agreement in which the end-user agrees to certain requirements for using the distributed product such as accepting distributor charges for content use and agreeing to observe the copyright rights of the creator. A third agreement might exist between the distributor and a financial clearinghouse that allows the distributor to employ the clearinghouse's credit for payment for the product if the end-user has a separate (fourth) agreement directly with the clearinghouse extending credit to the end-user. A fifth, evolving

10

15

20

agreement may develop between all value chain participants as content control information passes along its chain of handling. This evolving agreement can establish the rights of all parties to content usage information, including, for example, the nature of information to be received by each party and the pathway of handling of content usage information and related procedures. A sixth agreement in this example, may involve all parties to the agreement and establishes certain general assumptions, such as security techniques and degree of trustedness (for example, commercial integrity of the system may require each VDE installation secure subsystem to electronically warrant that their VDE node meets certain interoperability requirements). In the above example, these six agreements could comprise agreements of an extended agreement for this commercial value chain instance.

VDE agreements support evolving ("living") electronic agreement arrangements that can be modified by current and/or new participants through very simple to sophisticated "negotiations" between newly proposed content control information interacting with control information already in place and/or by negotiation between concurrently proposed content control information submitted by a plurality of parties. A given model may be asynchronously and progressively modified over

time in accordance with existing senior rules and such modification may be applied to all, to classes of, and/or to specific content, and/or to classes and/or specific users and/or user nodes. A given piece of content may be subject to different control information at different times or places of handling, depending on the evolution of its content control information (and/or on differing, applicable VDE installation content control information). The evolution of control information can occur during the passing along of one or more VDE control information containing objects, that is control information may be modified at one or more points along a chain of control information handling, so long as such modification is allowed. As a result, VDE managed content may have different control information applied at both different "locations" in a chain of content handling and at similar locations in differing chains of the handling of such content. Such different application of control information may also result from content control information specifying that a certain party or group of parties shall be subject to content control information that differs from another party or group of parties. For example, content control information for a given piece of content may be stipulated as senior information and therefore not changeable, might be put in place by a content creator and might stipulate that national distributors of a given piece of their content may be permitted to make 100,000 copies

per calendar quarter, so long as such copies are provided to boni
fide end-users, but may pass only a single copy of such content to
a local retailers and the control information limits such a retailer
to making no more than 1,000 copies per month for retail sales to
5 end-users. In addition, for example, an end-user of such content
might be limited by the same content control information to
making three copies of such content, one for each of three
different computers he or she uses (one desktop computer at
work, one for a desktop computer at home, and one for a portable
10 computer).

Electronic agreements supported by the preferred
embodiment of the present invention can vary from very simple
to very elaborate. They can support widely diverse information
15 management models that provide for electronic information
security, usage administration, and communication and may
support:

- 20 (a) secure electronic distribution of information, for
example commercial literary properties,
- (b) secure electronic information usage monitoring and
reporting,

- 5
- (c) secure financial transaction capabilities related to both electronic information and/or appliance usage and other electronic credit and/or currency usage and administration capabilities,
- (d) privacy protection for usage information a user does not wish to release, and
- 10 (e) "living" electronic information content dissemination models that flexibly accommodate:
- (1) a breadth of participants,
- 15 (2) one or more pathways (chains) for: the handling of content, content and/or appliance control information, reporting of content and/or appliance usage related information, and/or payment,
- 20 (3) supporting an evolution of terms and conditions incorporated into content control information, including use of electronic negotiation capabilities,

- (4) support the combination of multiple pieces of content to form new content aggregations, and
- (5) multiple concurrent models.

5

Secure Processing Units

An important part of VDE provided by the present invention is the core secure transaction control arrangement, herein called an SPU (or SPUs), that typically must be present in each user's computer, other electronic appliance, or network. SPUs provide a trusted environment for generating decryption keys, encrypting and decrypting information, managing the secure communication of keys and other information between electronic appliances (i.e. between VDE installations and/or between plural VDE instances within a single VDE installation), securely accumulating and managing audit trail, reporting, and budget information in secure and/or non-secure non-volatile memory, maintaining a secure database of control information management instructions, and providing a secure environment for performing certain other control and administrative functions.

A hardware SPU (rather than a software emulation) within a VDE node is necessary if a highly trusted environment

for performing certain VDE activities is required. Such a trusted environment may be created through the use of certain control software, one or more tamper resistant hardware modules such as a semiconductor or semiconductor chipset (including, for example, a tamper resistant hardware electronic appliance peripheral device), for use within, and/or operatively connected to, an electronic appliance. With the present invention, the trustedness of a hardware SPU can be enhanced by enclosing some or all of its hardware elements within tamper resistant packaging and/or by employing other tamper resisting techniques (e.g. microfusing and/or thin wire detection techniques). A trusted environment of the present invention implemented, in part, through the use of tamper resistant semiconductor design, contains control logic, such as a microprocessor, that securely executes VDE processes.

A VDE node's hardware SPU is a core component of a VDE secure subsystem and may employ some or all of an electronic appliance's primary control logic, such as a microcontroller, microcomputer or other CPU arrangement. This primary control logic may be otherwise employed for non VDE purposes such as the control of some or all of an electronic appliance's non-VDE functions. When operating in a hardware SPU mode, said primary control logic must be sufficiently secure so as to protect

and conceal important VDE processes. For example, a hardware SPU may employ a host electronic appliance microcomputer operating in protected mode while performing VDE related activities, thus allowing portions of VDE processes to execute

5 with a certain degree of security. This alternate embodiment is in contrast to the preferred embodiment wherein a trusted environment is created using a combination of one or more tamper resistant semiconductors that are not part of said primary control logic. In either embodiment, certain control

10 information (software and parameter data) must be securely maintained within the SPU, and further control information can be stored externally and securely (e.g. in encrypted and tagged form) and loaded into said hardware SPU when needed. In many cases, and in particular with microcomputers, the

15 preferred embodiment approach of employing special purpose secure hardware for executing said VDE processes, rather than using said primary control logic, may be more secure and efficient. The level of security and tamper resistance required for trusted SPU hardware processes depends on the commercial

20 requirements of particular markets or market niches, and may vary widely.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features and advantages provided by the present invention(s) may be better and more completely understood by referring to the following detailed description of presently preferred example embodiments in connection with the drawings, of which:

FIGURE 1 illustrates an example of a "Virtual Distribution Environment" provided in accordance with a preferred example/embodiment of this invention;

FIGURE 1A is a more detailed illustration of an example of the "Information Utility" shown in FIGURE 1;

FIGURE 2 illustrates an example of a chain of handling and control;

FIGURE 2A illustrates one example of how rules and control information may persist from one participant to another in the Figure 2 chain of handling and control;

FIGURE 3 shows one example of different control information that may be provided;

FIGURE 4 illustrates examples of some different types of rules and/or control information;

FIGURES 5A and 5B show an example of an "object";

5

FIGURE 6 shows an example of a Secure Processing Unit ("SPU");

FIGURE 7 shows an example of an electronic appliance;

10

FIGURE 8 is a more detailed block diagram of an example of the electronic appliance shown in FIGURE 7;

FIGURE 9 is a detailed view of an example of the Secure Processing Unit (SPU) shown in FIGURES 6 and 8;

15

Figure 9A shows an example combined secure processing unit and control processing unit;

20

Figure 9B shows an example secure processing unit integrated with a standard CPU;

FIGURE 10 shows an example of a "Rights Operating System" ("ROS") architecture provided by the Virtual Distribution Environment;

5 FIGURES 11A-11C show examples of functional relationship(s) between applications and the Rights Operating System;

10 FIGURES 11D-11J show examples of "components" and "component assemblies";

FIGURE 12 is a more detailed diagram of an example of the Rights Operating System shown in FIGURE 10;

15 FIGURE 12A shows an example of how "objects" can be created;

20 FIGURE 13 is a detailed block diagram of an example the software architecture for a "protected processing environment" shown in FIGURE 12;

FIGURES 14A-14C are examples of SPU memory maps provided by the protected processing environment shown in FIGURE 13;

FIGURE 15 illustrates an example of how the channel services manager and load module execution manager of FIGURE 13 can support a channel;

5 FIGURE 15A is an example of a channel header and channel detail records shown in FIGURE 15;

10 FIGURE 15B is a flowchart of an example of program control steps that may be performed by the FIGURE 13 protected processing environment to create a channel;

FIGURE 16 is a block diagram of an example of a secure data base structure;

15 FIGURE 17 is an illustration of an example of a logical object structure;

FIGURE 18 shows an example of a stationary object structure;

20 FIGURE 19 shows an example of a traveling object structure;

FIGURE 20 shows an example of a content object structure;

5 FIGURE 21 shows an example of an administrative object structure;

FIGURE 22 shows an example of a method core structure;

FIGURE 23 shows an example of a load module structure;

10 FIGURE 24 shows an example of a User Data Element (UDE) and/or Method Data Element (MDE) structure;

FIGURES 25A-25C show examples of "map meters";

15 FIGURE 26 shows an example of a permissions record (PERC) structure;

20 FIGURES 26A and 26B together show a more detailed example of a permissions record structure;

FIGURE 27 shows an example of a shipping table structure;

FIGURE 28 shows an example of a receiving table structure;

5 FIGURE 29 shows an example of an administrative event log structure;

FIGURE 30 shows an example inter-relationship between and use of the object registration table, subject table and user rights table shown in the FIGURE 16 secure database;

10

FIGURE 31 is a more detailed example of an object registration table shown in FIGURE 16;

FIGURE 32 is a more detailed example of subject table shown in FIGURE 16;

15

FIGURE 33 is a more detailed example of a user rights table shown in FIGURE 16;

20 FIGURE 34 shows a specific example of how a site record table and group record table may track portions of the secure database shown in FIGURE 16;

FIGURE 34A is an example of a FIGURE 34 site record
table structure;

FIGURE 34B is an example of a FIGURE 34 group record
5 table structure;

FIGURE 35 shows an example of a process for updating
the secure database;

10 FIGURE 36 shows an example of how new elements may
be inserted into the FIGURE 16 secure data base;

FIGURE 37 shows an example of how an element of the
secure database may be accessed;

15 FIGURE 38 is a flowchart example of how to protect a
secure database element;

FIGURE 39 is a flowchart example of how to back up a
20 secure database;

FIGURE 40 is a flowchart example of how to recover a
secure database from a backup;

FIGURES 41A-41D are a set of examples showing how a "chain of handling and control" may be enabled using "reciprocal methods";

5 FIGURES 42A-42D show an example of a "reciprocal" BUDGET method;

FIGURES 43A-43D show an example of a "reciprocal" REGISTER method;

10

FIGURES 44A-44C show an example of a "reciprocal" AUDIT method;

15 FIGURES 45-48 show examples of several methods being used together to control release of content or other information;

FIGURES 49, 49A-49F show an example OPEN method;

20 FIGURES 50, 50A-50F show an example of a READ method;

FIGURES 51, 51A-51F show an example of a WRITE method;

FIGURE 52 shows an example of a CLOSE method;

FIGURES 53A-53B show an example of an EVENT
method;

5

FIGURE 53C shows an example of a BILLING method;

FIGURE 54 shows an example of an ACCESS method;

10

FIGURES 55A-55B show examples of DECRYPT and
ENCRYPT methods;

FIGURE 56 shows an example of a CONTENT method;

15

FIGURES 57A and 57B show examples of EXTRACT and
EMBED methods;

FIGURE 58A shows an example of an OBSCURE method;

20

FIGURES 58B, 58C show examples of a FINGERPRINT
method;

FIGURE 59 shows an example of a DESTROY method;

FIGURE 60 shows an example of a PANIC method;

FIGURE 61 shows an example of a METER method;

5 FIGURE 62 shows an example of a key "convolution"
process;

FIGURE 63 shows an example of how different keys may
be generated using a key convolution process to determine a
10 "true" key;

FIGURES 64 and 65 show an example of how protected
processing environment keys may be initialized;

15 FIGURES 66 and 67 show example processes for
decrypting information contained within stationary and
traveling objects, respectively;

Figures 67A and 67B show example techniques for
20 cracking a software-based protected processing environment;

FIGURE 68 shows an example of how a protected
processing environment may be initialized;

FIGURE 69 shows an example of how firmware may be downloaded into a protected processing environment;

5 Figure 69A shows an example technique for distributing protected processing environment software;

 Figure 69B-69C show an example installation routine for installing a software-based protected processing environment;

10 Figure 69D shows example techniques for embedding cryptographic keys at random locations within structure-based protected processing environment operational materials;

15 Figure 69E shows example locations for PPE operational materials random modifications and/or digital fingerprints;

 Figure 69F shows an example customized static storage layout for PPE operational materials;

20 Figure 69G shows example electronic appliance signature locations;

 Figure 69H shows example sequence dependent and independent processes;

Figures 69I and 69J show example static code and data storage organizations;

5 Figures 69K-69L together show example steps for providing dynamic protection mechanisms;

Figure 69M shows an example initialization time check routine;

10 Figure 69N shows an example time check routine;

Figure 69O shows example time check data structures;

15 FIGURE 70 shows an example of multiple VDE electronic appliances connected together with a network or other communications means;

20 Figure 70A shows how content may be prepared for printing and encrypted inside a PPE, then decrypted inside a printer;

Figure 70B shows how characters may be selected from slightly different fonts in order to place an electronic fingerprint or watermark into printed output;

Figure 70C shows how characters in a font may be permuted to render a printed page unusable without the corresponding scrambled font;

5 FIGURE 71 shows an example of a portable VDE electronic appliance;

10 FIGURES 72A-72D show examples of "pop-up" displays that may be generated by the user notification and exception interface;

FIGURE 73 shows an example of a "smart object";

15 FIGURE 74 shows an example of a process using "smart objects";

FIGURES 75A-75D show examples of data structures used for electronic negotiation;

20 FIGURES 75E-75F show example structures relating to an electronic agreement;

FIGURES 76A-76B show examples of electronic negotiation processes;

FIGURE 77 shows a further example of a chain of handling and control;

FIGURE 78 shows an example of a VDE "repository";

5

FIGURES 79-83 show an example illustrating a chain of handling and control to evolve and transform VDE managed content and control information;

10

FIGURE 84 shows a further example of a chain of handling and control involving several categories of VDE participants;

15

FIGURE 85 shows a further example of a chain of distribution and handling within an organization;

Figures 86 and 86A show a further example of a chain of handling and control; and

20

Figure 87 shows an example of a virtual silicon container model.

MORE DETAILED DESCRIPTION

Figures 1-7 and the discussion below provides an overview of some aspects of features provided by this invention. Following this overview is a more technical "detail description" of example embodiments in accordance with the invention.

5

Overview

Figure 1 shows a "Virtual Distribution Environment" ("VDE") 100 that may be provided in accordance with this invention. In Figure 1, an information utility 200 connects to communications means 202 such as telephone or cable TV lines for example. Telephone or cable TV lines 202 may be part of an "electronic highway" that carries electronic information from place to place. Lines 202 connect information utility 200 to other people such as for example a consumer 208, an office 210, a video production studio 204, and a publishing house 214. Each of the people connected to information utility 200 may be called a "VDE participant" because they can participate in transactions occurring within the virtual distribution environment 100.

15
20

Almost any sort of transaction you can think of can be supported by virtual distribution environment 100. A few of many examples of transactions that can be supported by virtual distribution environment 100 include:

- home banking and electronic payments;
- electronic legal contracts;
- distribution of "content" such as electronic printed matter, video, audio, images and computer programs; and
- 5 • secure communication of private information such as medical records and financial information.

Virtual distribution environment 100 is "virtual" because it does not require many of the physical "things" that used to be necessary to protect rights, ensure reliable and predictable
10 distribution, and ensure proper compensation to content creators and distributors. For example, in the past, information was distributed on records or disks that were difficult to copy. In the past, private or secret content was distributed in sealed
15 envelopes or locked briefcases delivered by courier. To ensure appropriate compensation, consumers received goods and services only after they handed cash over to a seller. Although information utility 200 may deliver information by transferring physical "things" such as electronic storage media, the virtual
20 distribution environment 100 facilitates a completely electronic "chain of handling and control."

VDE Flexibility Supports Transactions

Information utility 200 flexibly supports many different kinds of information transactions. Different VDE participants may define and/or participate in different parts of a transaction. Information utility 200 may assist with delivering information about a transaction, or it may be one of the transaction participants.

For example, the video production studio 204 in the upper right-hand corner of Figure 1 may create video/television programs. Video production studio 204 may send these programs over lines 202, or may use other paths such as satellite link 205 and CD ROM delivery service 216. Video production studio 204 can send the programs directly to consumers 206, 208, 210, or it can send the programs to information utility 200 which may store and later send them to the consumers, for example. Consumers 206, 208, 210 are each capable of receiving and using the programs created by video production studio 204—assuming, that is, that the video production studio or information utility 200 has arranged for these consumers to have appropriate “rules and controls” (control information) that give the consumers rights to use the programs.

Even if a consumer has a copy of a video program, she cannot watch or copy the program unless she has “rules and

controls" that authorize use of the program. She can use the program only as permitted by the "rules and controls."

5 For example, video production studio 204 might release a half-hour exercise video in the hope that as many viewers as possible will view it. The video production studio 204 wishes to receive \$2.00 per viewing. Video production studio 204 may, through information utility 200, make the exercise video available in "protected" form to all consumers 206, 208, 210.

10 Video production studio 204 may also provide "rules and controls" for the video. These "rules and controls" may specify for example:

15 (1) any consumer who has good credit of at least \$2.00 based on a credit account with independent financial provider 212 (such as Mastercard or VISA) may watch the video,

20 (2) virtual distribution environment 100 will "meter" each time a consumer watches the video, and report usage to video production studio 204 from time to time, and

(3) financial provider 212 may electronically collect payment (\$2.00) from the credit account of each consumer

who watches the video, and transfer these payments to the video production studio 204.

Information utility 200 allows even a small video production studio to market videos to consumers and receive compensation for its efforts. Moreover, the videos can, with appropriate payment to the video production studio, be made available to other video publishers who may add value and/or act as repackagers or redistributors.

Figure 1 also shows a publishing house 214. Publishing house 214 may act as a distributor for an author 206. The publishing house 214 may distribute rights to use "content" (such as computer software, electronic newspapers, the video produced by publishing house 214, audio, or any other data) to consumers such as office 210. The use rights may be defined by "rules and controls" distributed by publishing house 216. Publishing house 216 may distribute these "rules and controls" with the content, but this is not necessary. Because the content can be used only by consumers that have the appropriate "rules and controls," content and its associated "rules and controls" may be distributed at different times, in different ways, by different VDE participants. The ability of VDE to securely distribute and

enforce "rules and controls" separately from the content they apply to provides great advantages.

5 Use rights distributed by publishing house 214 may, for example, permit office 210 to make and distribute copies of the content to its employees. Office 210 may act as a redistributor by extending a "chain of handling and control" to its employees. The office 210 may add or modify "rules and controls" (consistent with the "rules and controls" it receives from publishing house
10 214) to provide office-internal control information and mechanisms. For example, office 210 may set a maximum usage budget for each individual user and/or group within the office, or it may permit only specified employees and/or groups to access certain information.

15

 Figure 1 also shows an information delivery service 216 delivering electronic storage media such as "CD ROM" disks to consumers 206. Even though the electronic storage media themselves are not delivered electronically by information utility
20 200 over lines 202, they are still part of the virtual distribution environment 100. The electronic storage media may be used to distribute content, "rules and controls," or other information.

Example of What's Inside Information Utility 200

“Information utility” 200 in Figure 1 can be a collection of participants that may act as distributors, financial clearinghouses, and administrators. Figure 1A shows an example of what may be inside one example of information utility 200. Information utility participants 200a-200g could each be an independent organization/business. There can be any number of each of participants 200a-200g. In this example, electronic “switch” 200a connects internal parts of information utility 200 to each other and to outside participants, and may also connect outside participants to one another.

Information utility 200 may include a “transaction processor” 200b that processes transactions (to transfer electronic funds, for example) based on requests from participants and/or report receiver 200e. It may also include a “usage analyst” 200c that analyzes reported usage information. A “report creator” 200d may create reports based on usage for example, and may provide these reports to outside participants and/or to participants within information utility 200. A “report receiver” 200e may receive reports such as usage reports from content users. A “permissioning agent” 200f may distribute “rules and controls” granting usage or distribution permissions based on a profile of a consumer’s credit worthiness, for example.

An administrator 200h may provide information that keeps the virtual distribution environment 100 operating properly. A content and message storage 200g may store information for use by participants within or outside of information utility 200.

5

Example of Distributing Content" Using A Chain of Handling and Control"

As explained above, virtual distribution environment 100 can be used to manage almost any sort of transaction. One type of important transaction that virtual distribution environment 100 may be used to manage is the distribution or communication of "content" or other important information. Figure 2 more abstractly shows a "model" of how the Figure 1 virtual distribution environment 100 may be used to provide a "chain of handling and control" for distributing content. Each of the blocks in Figure 2 may correspond to one or more of the VDE participants shown in Figure 1.

10
15

In the Figure 2 example, a VDE content creator 102 creates "content." The content creator 102 may also specify "rules and controls" for distributing the content. These distribution-related "rules and controls" can specify who has permission to distribute the rights to use content, and how many users are allowed to use the content.

20
25

Arrow 104 shows the content creator 102 sending the
"rules and controls" associated with the content to a VDE rights
distributor 106 ("distributor") over an electronic highway 108 (or
by some other path such as an optical disk sent by a delivery
5 service such as U. S. mail). The content can be distributed over
the same or different path used to send the "rules and controls."
The distributor 106 generates her own "rules and controls" that
relate to usage of the content. The usage-related "rules and
controls" may, for example, specify what a user can and can't do
10 with the content and how much it costs to use the content. These
usage-related "rules and controls" must be consistent with the
"rules and controls" specified by content creator 102.

Arrow 110 shows the distributor 106 distributing rights to
15 use the content by sending the content's "rules and controls" to a
content user 112 such as a consumer. The content user 112 uses
the content in accordance with the usage-related "rules and
controls."

20 In this Figure 2 example, information relating to content
use is, as shown by arrow 114, reported to a financial
clearinghouse 116. Based on this "reporting," the financial
clearinghouse 116 may generate a bill and send it to the content
user 112 over a "reports and payments" network 118. Arrow 120

shows the content user 112 providing payments for content usage to the financial clearinghouse 116. Based on the reports and payments it receives, the financial clearinghouse 116 may provide reports and/or payments to the distributor 106. The distributor 106 may, as shown by arrow 122, provide reports and/or payments to the content creator 102. The clearinghouse 116 may provide reports and payments directly to the creator 102. Reporting and/or payments may be done differently. For example, clearinghouse 116 may directly or through an agent, provide reports and/or payments to each of VDE content creators 102, and rights distributor 106, as well as reports to content user 112.

The distributor 106 and the content creator 102 may be the same person, or they may be different people. For example, a musical performing group may act as both content creator 102 and distributor 106 by creating and distributing its own musical recordings. As another example, a publishing house may act as a distributor 106 to distribute rights to use works created by an author content creator 102. Content creators 102 may use a distributor 106 to efficiently manage the financial end of content distribution.

The "financial clearinghouse" 116 shown in Figure 2 may also be a "VDE administrator." Financial clearinghouse 116 in its VDE administrator role sends "administrative" information to the VDE participants. This administrative information helps to keep the virtual distribution environment 100 operating properly. The "VDE administrator" and financial clearinghouse roles may be performed by different people or companies, and there can be more than one of each.

10 **More about Rules and Controls"**

The virtual distribution environment 100 prevents use of protected information except as permitted by the "rules and controls" (control information). For example, the "rules and controls" shown in Figure 2 may grant specific individuals or classes of content users 112 "permission" to use certain content. They may specify what kinds of content usage are permitted, and what kinds are not. They may specify how content usage is to be paid for and how much it costs. As another example, "rules and controls" may require content usage information to be reported back to the distributor 106 and/or content creator 102.

Every VDE participant in "chain of handling and control" is normally subject to "rules and controls." "Rules and controls" define the respective rights and obligations of each of the various

VDE participants. "Rules and controls" provide information and mechanisms that may establish interdependencies and relationships between the participants. "Rules and controls" are flexible, and permit "virtual distribution environment" 100 to support most "traditional" business transactions. For example:

- "Rules and controls" may specify which financial clearinghouse(s) 116 may process payments,
- "Rules and controls" may specify which participant(s) receive what kind of usage report, and
- "Rules and controls" may specify that certain information is revealed to certain participants, and that other information is kept secret from them.

"Rules and controls" may self limit if and how they may be changed. Often, "rules and controls" specified by one VDE participant cannot be changed by another VDE participant. For example, a content user 112 generally can't change "rules and controls" specified by a distributor 106 that require the user to pay for content usage at a certain rate. "Rules and controls" may "persist" as they pass through a "chain of handling and control," and may be "inherited" as they are passed down from one VDE participant to the next.

Depending upon their needs, VDE participants can specify that their "rules and controls" can be changed under conditions specified by the same or other "rules and controls." For example, "rules and controls" specified by the content creator 102 may
5 permit the distributor 106 to "mark up" the usage price just as retail stores "mark up" the wholesale price of goods. Figure 2A shows an example in which certain "rules and controls" persist unchanged from content creator 102 to content user 112; other "rules and controls" are modified or deleted by distributor 106;
10 and still other "rules and controls" are added by the distributor.

"Rules and controls" can be used to protect the content user's privacy by limiting the information that is reported to other VDE participants. As one example, "rules and controls"
15 can cause content usage information to be reported anonymously without revealing content user identity, or it can reveal only certain information to certain participants (for example, information derived from usage) with appropriate permission, if required. This ability to securely control what information is
20 revealed and what VDE participant(s) it is revealed to allows the privacy rights of all VDE participants to be protected.

Rules and Contents" Can Be Separately Delivered

As mentioned above, virtual distribution environment 100 "associates" content with corresponding "rules and controls," and prevents the content from being used or accessed unless a set of corresponding "rules and controls" is available. The distributor 106 doesn't need to deliver content to control the content's distribution. The preferred embodiment can securely protect content by protecting corresponding, usage enabling "rules and controls" against unauthorized distribution and use.

In some examples, "rules and controls" may travel with the content they apply to. Virtual distribution environment 100 also allows "rules and controls" to be delivered separately from content. Since no one can use or access protected content without "permission" from corresponding "rules and controls," the distributor 106 can control use of content that has already been (or will in the future be) delivered. "Rules and controls" may be delivered over a path different from the one used for content delivery. "Rules and controls" may also be delivered at some other time. The content creator 102 might deliver content to content user 112 over the electronic highway 108, or could make the content available to anyone on the highway. Content may be used at the time it is delivered, or it may be stored for later use or reuse.

The virtual distribution environment 100 also allows payment and reporting means to be delivered separately. For example, the content user 112 may have a virtual "credit card" that extends credit (up to a certain limit) to pay for usage of any content. A "credit transaction" can take place at the user's site without requiring any "online" connection or further authorization. This invention can be used to help securely protect the virtual "credit card" against unauthorized use.

Rules and Contents" Define Processes

Figure 3 shows an example of an overall process based on "rules and controls." It includes an "events" process 402, a meter process 404, a billing process 406, and a budget process 408. Not all of the processes shown in Figure 3 will be used for every set of "rules and controls."

The "events process" 402 detects things that happen ("events") and determines which of those "events" need action by the other "processes." The "events" may include, for example, a request to use content or generate a usage permission. Some events may need additional processing, and others may not. Whether an "event" needs more processing depends on the "rules and controls" corresponding to the content. For example, a user who lacks permission will not have her request satisfied ("No

Go"). As another example, each user request to turn to a new page of an electronic book may be satisfied ("Go"), but it may not be necessary to meter, bill or budget those requests. A user who has purchased a copy of a novel may be permitted to open and
5 read the novel as many times as she wants to without any further metering, billing or budgeting. In this simple example, the "event process" 402 may request metering, billing and/or budgeting processes the first time the user asks to open the protected novel (so the purchase price can be charged to the
10 user), and treat all later requests to open the same novel as "insignificant events." Other content (for example, searching an electronic telephone directory) may require the user to pay a fee for each access.

15 "Meter" process 404 keeps track of events, and may report usage to distributor 106 and/or other appropriate VDE participant(s). Figure 4 shows that process 404 can be based on a number of different factors such as:

- (a) type of usage to charge for,
- 20 (b) what kind of unit to base charges on,
- (c) how much to charge per unit,

(d) when to report, and

(e) how to pay.

These factors may be specified by the "rules and controls" that control the meter process.

5

Billing process 406 determines how much to charge for events. It records and reports payment information.

10 Budget process 408 limits how much content usage is permitted. For example, budget process 408 may limit the number of times content may be accessed or copied, or it may limit the number of pages or other amount of content that can be used based on, for example, the number of dollars available in a credit account. Budget process 408 records and reports financial
15 and other transaction information associated with such limits.

Content may be supplied to the user once these processes have been successfully performed.

20 **Containers and Objects"**

Figure 5A shows how the virtual distribution environment 100, in a preferred embodiment, may package information elements (content) into a "container" 302 so the information can't be accessed except as provided by its "rules and controls."

Normally, the container 302 is electronic rather than physical. Electronic container 302 in one example comprises "digital" information having a well defined structure. Container 302 and its contents can be called an "object 300."

5

The Figure 5A example shows items "within" and enclosed by container 302. However, container 302 may "contain" items without those items actually being stored within the container. For example, the container 302 may reference items that are available elsewhere such as in other containers at remote sites. Container 302 may reference items available at different times or only during limited times. Some items may be too large to store within container 302. Items may, for example, be delivered to the user in the form of a "live feed" of video at a certain time. Even then, the container 302 "contains" the live feed (by reference) in this example.

10

15

20

Container 302 may contain information content 304 in electronic (such as "digital") form. Information content 304 could be the text of a novel, a picture, sound such as a musical performance or a reading, a movie or other video, computer software, or just about any other kind of electronic information you can think of. Other types of "objects" 300 (such as

"administrative objects") may contain "administrative" or other information instead of or in addition to information content 304.

In the Figure 5A example, container 302 may also contain
5 "rules and controls" in the form of:

- (a) a "permissions record" 808;
- (b) "budgets" 308; and
- (c) "other methods" 1000.

10 Figure 5B gives some additional detail about permissions record 808, budgets 308 and other methods 1000. The "permissions record" 808 specifies the rights associated with the object 300 such as, for example, who can open the container 302, who can use the object's contents, who can distribute the object,
15 and what other control mechanisms must be active. For example, permissions record 808 may specify a user's rights to use, distribute and/or administer the container 302 and its content. Permissions record 808 may also specify requirements to be applied by the budgets 308 and "other methods" 1000.
20 Permissions record 808 may also contain security related information such as scrambling and descrambling "keys."

"Budgets" 308 shown in Figure 5B are a special type of "method" 1000 that may specify, among other things, limitations

on usage of information content 304, and how usage will be paid for. Budgets 308 can specify, for example, how much of the total information content 304 can be used and/or copied. The methods 310 may prevent use of more than the amount specified by a specific budget.

“Other methods” 1000 define basic operations used by “rules and controls.” Such “methods” 1000 may include, for example, how usage is to be “metered,” if and how content 304 and other information is to be scrambled and descrambled, and other processes associated with handling and controlling information content 304. For example, methods 1000 may record the identity of anyone who opens the electronic container 302, and can also control how information content is to be charged based on “metering.” Methods 1000 may apply to one or several different information contents 304 and associated containers 302, as well as to all or specific portions of information content 304.

Secure Processing Unit (SPU)

The “VDE participants” may each have an “electronic appliance.” The appliance may be or contain a computer. The appliances may communicate over the electronic highway 108. Figure 6 shows a secure processing unit (“SPU”) 500 portion of

the "electronic appliance" used in this example by each VDE participant. SPU 500 processes information in a secure processing environment 503, and stores important information securely. SPU 500 may be emulated by software operating in a
5 host electronic appliance.

SPU 500 is enclosed within and protected by a "tamper resistant security barrier" 502. Security barrier 502 separates the secure environment 503 from the rest of the world. It
10 prevents information and processes within the secure environment 503 from being observed, interfered with and leaving except under appropriate secure conditions. Barrier 502 also controls external access to secure resources, processes and information within SPU 500. In one example, tamper resistant
15 security barrier 502 is formed by security features such as "encryption," and hardware that detects tampering and/or destroys sensitive information within secure environment 503
when tampering is detected.

20 SPU 500 in this example is an integrated circuit ("IC") "chip" 504 including "hardware" 506 and "firmware" 508. SPU 500 connects to the rest of the electronic appliance through an "appliance link" 510. SPU "firmware" 508 in this example is "software" such as a "computer program(s)" "embedded" within

chip 504. Firmware 508 makes the hardware 506 work.

Hardware 506 preferably contains a processor to perform instructions specified by firmware 508. "Hardware" 506 also contains long-term and short-term memories to store information securely so it can't be tampered with. SPU 500 may also have a protected clock/calendar used for timing events. The SPU hardware 506 in this example may include special purpose electronic circuits that are specially designed to perform certain processes (such as "encryption" and "decryption") rapidly and efficiently.

The particular context in which SPU 500 is being used will determine how much processing capabilities SPU 500 should have. SPU hardware 506, in this example, provides at least enough processing capabilities to support the secure parts of processes shown in Figure 3. In some contexts, the functions of SPU 500 may be increased so the SPU can perform all the electronic appliance processing, and can be incorporated into a general purpose processor. In other contexts, SPU 500 may work alongside a general purpose processor, and therefore only needs to have enough processing capabilities to handle secure processes.

VDE Electronic Appliance and Rights Operating System"

Figure 7 shows an example of an electronic appliance 600 including SPU 500. Electronic appliance 600 may be practically any kind of electrical or electronic device, such as:

5

- a computer
- a T.V. "set top" control box
- a pager
- a telephone
- 10 • a sound system
- a video reproduction system
- a video game player
- a "smart" credit card

15

Electronic appliance 600 in this example may include a keyboard or keypad 612, a voice recognizer 613, and a display 614. A human user can input commands through keyboard 612 and/or voice recognizer 613, and may view information on display 614.

20

Appliance 600 may communicate with the outside world through any of the connections/devices normally used within an electronic appliance. The connections/devices shown along the bottom of the drawing are examples:

a "modem" 618 or other telecommunications link;

a CD ROM disk 620 or other storage medium or device;

a printer 622;
broadcast reception 624;
a document scanner 626; and
a "cable" 628 connecting the appliance with a "network."

5

Virtual distribution environment 100 provides a "rights operating system" 602 that manages appliance 600 and SPU 500 by controlling their hardware resources. The operating system 602 may also support at least one "application" 608. Generally,
10 "application" 608 is hardware and/or software specific to the context of appliance 600. For example, if appliance 600 is a personal computer, then "application" 608 could be a program loaded by the user, for instance, a word processor, a communications system or a sound recorder. If appliance 600 is
15 a television controller box, then application 608 might be hardware or software that allows a user to order videos on demand and perform other functions such as fast forward and rewind. In this example, operating system 602 provides a standardized, well defined, generalized "interface" that could
20 support and work with many different "applications" 608.

Operating system 602 in this example provides "rights and auditing operating system functions" 604 and "other operating system functions" 606. The "rights and auditing operating

system functions" 604 securely handle tasks that relate to virtual distribution environment 100. SPU 500 provides or supports many of the security functions of the "rights and auditing operating system functions" 402. The "other operating system functions" 606 handle general appliance functions. Overall operating system 602 may be designed from the beginning to include the "rights and auditing operating system functions" 604 plus the "other operating system functions" 606, or the "rights and auditing operating system functions" may be an add-on to a preexisting operating system providing the "other operating system functions."

"Rights operating system" 602 in this example can work with many different types of appliances 600. For example, it can work with large mainframe computers, "minicomputers" and "microcomputers" such as personal computers and portable computing devices. It can also work in control boxes on the top of television sets, small portable "pagers," desktop radios, stereo sound systems, telephones, telephone switches, or any other electronic appliance. This ability to work on big appliances as well as little appliances is called "scalable." A "scalable" operating system 602 means that there can be a standardized interface across many different appliances performing a wide variety of tasks.

The "rights operating system functions" 604 are "services-based" in this example. For example, "rights operating system functions" 604 handle summary requests from application 608 rather than requiring the application to always make more detailed "subrequests" or otherwise get involved with the underlying complexities involved in satisfying a summary request. For example, application 608 may simply ask to read specified information; "rights operating system functions" 604 can then decide whether the desired information is VDE-protected content and, if it is, perform processes needed to make the information available. This feature is called "transparency." "Transparency" makes tasks easy for the application 608. "Rights operating system functions" 604 can support applications 608 that "know" nothing about virtual distribution environment 100. Applications 608 that are "aware" of virtual distribution environment 100 may be able to make more detailed use of virtual distribution environment 100.

In this example, "rights operating system functions" 604 are "event driven". Rather than repeatedly examining the state of electronic appliance 600 to determine whether a condition has arisen, the "rights operating system functions" 604 may respond directly to "events" or "happenings" within appliance 600.

In this example, some of the services performed by "rights operating system functions" 604 may be extended based on additional "components" delivered to operating system 602.

"Rights operating system functions" 604 can collect together and

5 use "components" sent by different participants at different times. The "components" help to make the operating system 602 "scalable." Some components can change how services work on little appliances versus how they work on big appliances (e.g., multi-user). Other components are designed to work with
10 specific applications or classes of applications (e.g., some types of meters and some types of budgets).

Electronic Appliance 600

15 An electronic appliance 600 provided by the preferred embodiment may, for example, be any electronic apparatus that contains one or more microprocessors and/or microcontrollers and/or other devices which perform logical and/or mathematical calculations. This may include computers; computer terminals; device controllers for use with computers; peripheral devices for
20 use with computers; digital display devices; televisions; video and audio/video projection systems; channel selectors and/or decoders for use with broadcast and/or cable transmissions; remote control devices; video and/or audio recorders; media players including compact disc players, videodisc players and

tape players; audio and/or video amplifiers; virtual reality machines; electronic game players; multimedia players; radios; telephones; videophones; facsimile machines; robots; numerically controlled machines including machine tools and the like; and
5 other devices containing one or more microcomputers and/or microcontrollers and/or other CPUs, including those not yet in existence.

Figure 8 shows an example of an electronic appliance 600.
10 This example of electronic appliance 600 includes a system bus 653. In this example, one or more conventional general purpose central processing units ("CPUs") 654 are connected to bus 653. Bus 653 connects CPU(s) 654 to RAM 656, ROM 658, and I/O controller 660. One or more SPUs 500 may also be connected to
15 system bus 653. System bus 653 may permit SPU(s) 500 to communicate with CPU(s) 654, and also may allow both the CPU(s) and the SPU(s) to communicate (e.g., over shared address and data lines) with RAM 656, ROM 658 and I/O controller 660. A power supply 659 may provide power to SPU
20 500, CPU 654 and the other system components shown.

In the example shown, I/O controller 660 is connected to secondary storage device 652, a keyboard/display 612,614, a communications controller 666, and a backup storage device 668.

Backup storage device 668 may, for example, store information on mass media such as a tape 670, a floppy disk, a removable memory card, etc. Communications controller 666 may allow electronic appliance 600 to communicate with other electronic appliances via network 672 or other telecommunications links. Different electronic appliances 600 may interoperate even if they use different CPUs and different instances of ROS 602, so long as they typically use compatible communication protocols and/or security methods. In this example, I/O controller 660 permits CPU 654 and SPU 500 to read from and write to secondary storage 662, keyboard/display 612, 614, communications controller 666, and backup storage device 668.

Secondary storage 662 may comprise the same one or more non-secure secondary storage devices (such as a magnetic disk and a CD-ROM drive as one example) that electronic appliance 600 uses for general secondary storage functions. In some implementations, part or all of secondary storage 652 may comprise a secondary storage device(s) that is physically enclosed within a secure enclosure. However, since it may not be practical or cost-effective to physically secure secondary storage 652 in many implementations, secondary storage 652 may be used to store information in a secure manner by encrypting information before storing it in secondary storage 652. If information is

encrypted before it is stored, physical access to secondary storage 652 or its contents does not readily reveal or compromise the information.

5 Secondary storage 652 in this example stores code and data used by CPU 654 and/or SPU 500 to control the overall operation of electronic appliance 600. For example, Figure 8 shows that "Rights Operating System" ("ROS") 602 (including a portion 604 of ROS that provides VDE functions and a portion 10 606 that provides other OS functions) shown in Figure 7 may be stored on secondary storage 652. Secondary storage 652 may also store one or more VDE objects 300. Figure 8 also shows that the secure files 610 shown in Figure 7 may be stored on secondary storage 652 in the form of a "secure database" or 15 management file system 610. This secure database 610 may store and organize information used by ROS 602 to perform VDE functions 604. Thus, the code that is executed to perform VDE and other OS functions 604, 606, and secure files 610 (as well as VDE objects 300) associated with those functions may be stored 20 in secondary storage 652. Secondary storage 652 may also store "other information" 673 such as, for example, information used by other operating system functions 606 for task management, non-VDE files, etc. Portions of the elements indicated in secondary storage 652 may also be stored in ROM 658, so long as

those elements do not require changes (except when ROM 658 is replaced). Portions of ROS 602 in particular may desirably be included in ROM 658 (e.g., "bootstrap" routines, POST routines, etc. for use in establishing an operating environment for
5 electronic appliance 600 when power is applied).

Figure 8 shows that secondary storage 652 may also be used to store code ("application programs") providing user application(s) 608 shown in Figure 7. Figure 8 shows that there
10 may be two general types of application programs 608: "VDE aware" applications 608a, and Non-VDE aware applications 608b. VDE aware applications 608a may have been at least in part designed specifically with VDE 100 in mind to access and take detailed advantage of VDE functions 604. Because of the
15 "transparency" features of ROS 602, non-VDE aware applications 608b (e.g., applications not specifically designed for VDE 100) can also access and take advantage of VDE functions 604.

20 **SECURE PROCESSING UNIT 500**

Each VDE node or other electronic appliance 600 in the preferred embodiment may include one or more SPUs 500. SPUs 500 may be used to perform all secure processing for VDE 100. For example, SPU 500 is used for decrypting (or otherwise

unsecuring) VDE protected objects 300. It is also used for managing encrypted and/or otherwise secured communication (such as by employing authentication and/or error-correction validation of information). SPU 500 may also perform secure data management processes including governing usage of, auditing of, and where appropriate, payment for VDE objects 300 (through the use of prepayments, credits, real-time electronic debits from bank accounts and/or VDE node currency token deposit accounts). SPU 500 may perform other transactions related to such VDE objects 300.

SPU Physical Packaging and Security Barrier 502

As shown Figure 6, in the preferred embodiment, an SPU 500 may be implemented as a single integrated circuit "chip" 505 to provide a secure processing environment in which confidential and/or commercially valuable information can be safely processed, encrypted and/or decrypted. IC chip 505 may, for example, comprise a small semiconductor "die" about the size of a thumbnail. This semiconductor die may include semiconductor and metal conductive pathways. These pathways define the circuitry, and thus the functionality, of SPU 500. Some of these pathways are electrically connected to the external "pins" 504 of the chip 505.

As shown in Figures 6 and 9, SPU 500 may be surrounded by a tamper-resistant hardware security barrier 502. Part of this security barrier 502 is formed by a plastic or other package in which an SPU "die" is encased. Because the processing occurring within, and information stored by, SPU 500 are not easily accessible to the outside world, they are relatively secure from unauthorized access and tampering. All signals cross barrier 502 through a secure, controlled path provided by BIU 530 that restricts the outside world's access to the internal components within SPU 500. This secure, controlled path resists attempts from the outside world to access secret information and resources within SPU 500.

It is possible to remove the plastic package of an IC chip and gain access to the "die." It is also possible to analyze and "reverse engineer" the "die" itself (e.g., using various types of logic analyzers and microprobes to collect and analyze signals on the die while the circuitry is operating, using acid etching or other techniques to remove semiconductor layers to expose other layers, viewing and photographing the die using an electron microscope, etc.) Although no system or circuit is absolutely impervious to such attacks, SPU barrier 502 may include additional hardware protections that make successful attacks exceedingly costly and time consuming. For example, ion

implantation and/or other fabrication techniques may be used to make it very difficult to visually discern SPU die conductive pathways, and SPU internal circuitry may be fabricated in such a way that it "self-destructs" when exposed to air and/or light.

5 SPU 500 may store secret information in internal memory that loses its contents when power is lost. Circuitry may be incorporated within SPU 500 that detects microprobing or other tampering, and self-destructs (or destroys other parts of the SPU) when tampering is detected. These and other hardware-
10 based physical security techniques contribute to tamper-resistant hardware security barrier 502.

To increase the security of security barrier 502 even further, it is possible to encase or include SPU 500 in one or
15 more further physical enclosures such as, for example: epoxy or other "potting compound"; further module enclosures including additional self-destruct, self-disabling or other features activated when tampering is detected; further modules providing additional security protections such as requiring password or
20 other authentication to operate; and the like. In addition, further layers of metal may be added to the die to complicate acid etching, micro probing, and the like; circuitry designed to "zeroize" memory may be included as an aspect of self-destruct processes; the plastic package itself may be designed to resist

chemical as well as physical "attacks"; and memories internal to SPU 500 may have specialized addressing and refresh circuitry that "shuffles" the location of bits to complicate efforts to electrically determine the value of memory locations. These and
5 other techniques may contribute to the security of barrier 502.

In some electronic appliances 600, SPU 500 may be integrated together with the device microcontroller or equivalent or with a device I/O or communications microcontroller into a
10 common chip (or chip set) 505. For example, in one preferred embodiment, SPU 500 may be integrated together with one or more other CPU(s) (e.g., a CPU 654 of an electronic appliance) in a single component or package. The other CPU(s) 654 may be any centrally controlling logic arrangement, such as for example,
15 a microprocessor, other microcontroller, and/or array or other parallel processor. This integrated configuration may result in lower overall cost, smaller overall size, and potentially faster interaction between an SPU 500 and a CPU 654. Integration may also provide wider distribution if an integrated SPU/CPU
20 component is a standard feature of a widely distributed microprocessor line. Merging an SPU 500 into a main CPU 654 of an electronic appliance 600 (or into another appliance or appliance peripheral microcomputer or other microcontroller) may substantially reduce the overhead cost of implementing

VDE 100. Integration considerations may include cost of implementation, cost of manufacture, desired degree of security, and value of compactness.

5 SPU 500 may also be integrated with devices other than CPUs. For example, for video and multimedia applications, some performance and/or security advantages (depending on overall design) could result from integrating an SPU 500 into a video controller chip or chipset. SPU 500 can also be integrated
10 directly into a network communications chip or chipset or the like. Certain performance advantages in high speed communications applications may also result from integrating an SPU 500 with a modem chip or chipset. This may facilitate incorporation of an SPU 500 into communication appliances such
15 as stand-alone fax machines. SPU 500 may also be integrated into other peripheral devices, such as CD-ROM devices, set-top cable devices, game devices, and a wide variety of other electronic appliances that use, allow access to, perform transactions related to, or consume, distributed information.

20

SPU 500 Internal Architecture

Figure 9 is a detailed diagram of the internal structure within an example of SPU 500. SPU 500 in this example includes a single microprocessor 520 and a limited amount of

memory configured as ROM 532 and RAM 534. In more detail,
this example of SPU 500 includes microprocessor 520, an
encrypt/decrypt engine 522, a DMA controller 526, a real-time
clock 528, a bus interface unit ("BIU") 530, a read only memory
5 (ROM) 532, a random access memory (RAM) 534, and a memory
management unit ("MMU") 540. DMA controller 526 and MMU
540 are optional, but the performance of SPU 500 may suffer if
they are not present. SPU 500 may also include an optional
pattern matching engine 524, an optional random number
10 generator 542, an optional arithmetic accelerator circuit 544, and
optional compression/decompression circuit 546. A shared
address/data bus arrangement 536 may transfer information
between these various components under control of
microprocessor 520 and/or DMA controller 526. Additional or
15 alternate dedicated paths 538 may connect microprocessor 520 to
the other components (e.g., encrypt/decrypt engine 522 via line
538a, real-time clock 528 via line 538b, bus interface unit 530 via
line 538c, DMA controller via line 538d, and memory
management unit (MMU) 540 via line 538e).

20
The following section discusses each of these SPU
components in more detail.

Microprocessor 520

Microprocessor 520 is the "brain" of SPU 500. In this example, it executes a sequence of steps specified by code stored (at least temporarily) within ROM 532 and/or RAM 534.

5 Microprocessor 520 in the preferred embodiment comprises a dedicated central processing arrangement (e.g., a RISC and/or CISC processor unit, a microcontroller, and/or other central processing means or, less desirably in most applications, process specific dedicated control logic) for executing instructions stored
10 in the ROM 532 and/or other memory. Microprocessor 520 may be separate elements of a circuitry layout, or may be separate packages within a secure SPU 500.

In the preferred embodiment, microprocessor 520 normally
15 handles the most security sensitive aspects of the operation of electronic appliance 600. For example, microprocessor 520 may manage VDE decrypting, encrypting, certain content and/or appliance usage control information, keeping track of usage of VDE secured content, and other VDE usage control related
20 functions.

Stored in each SPU 500 and/or electronic appliance secondary memory 652 may be, for example, an instance of ROS 602 software, application programs 608, objects 300 containing

VDE controlled property content and related information, and management database 610 that stores both information associated with objects and VDE control information. ROS 602 includes software intended for execution by SPU microprocessor 520 for, in part, controlling usage of VDE related objects 300 by electronic appliance 600. As will be explained, these SPU programs include "load modules" for performing basic control functions. These various programs and associated data are executed and manipulated primarily by microprocessor 520.

Real Time Clock (RTC) 528

In the preferred embodiment, SPU 500 includes a real time clock circuit ("RTC") 528 that serves as a reliable, tamper resistant time base for the SPU. RTC 528 keeps track of time of day and date (e.g., month, day and year) in the preferred embodiment, and thus may comprise a combination calendar and clock. A reliable time base is important for implementing time based usage metering methods, "time aged decryption keys," and other time based SPU functions.

The RTC 528 must receive power in order to operate. Optimally, the RTC 528 power source could comprise a small battery located within SPU 500 or other secure enclosure. However, the RTC 528 may employ a power source such as an

externally located battery that is external to the SPU 500. Such
an externally located battery may provide relatively
uninterrupted power to RTC 528, and may also maintain as
non-volatile at least a portion of the otherwise volatile RAM 534
5 within SPU 500.

In one implementation, electronic appliance power supply
659 is also used to power SPU 500. Using any external power
supply as the only power source for RTC 528 may significantly
10 reduce the usefulness of time based security techniques unless,
at minimum, SPU 500 recognizes any interruption (or any
material interruption) of the supply of external power, records
such interruption, and responds as may be appropriate such as
disabling the ability of the SPU 500 to perform certain or all
15 VDE processes. Recognizing a power interruption may, for
example, be accomplished by employing a circuit which is
activated by power failure. The power failure sensing circuit
may power another circuit that includes associated logic for
recording one or more power fail events. Capacitor discharge
20 circuitry may provide the necessary temporary power to operate
this logic. In addition or alternatively, SPU 500 may from time
to time compare an output of RTC 528 to a clock output of a host
electronic appliance 600, if available. In the event a discrepancy
is detected, SPU 500 may respond as appropriate, including

recording the discrepancy and/or disabling at least some portion of processes performed by SPU 500 under at least some circumstances.

5 If a power failure and/or RTC 528 discrepancy and/or other event indicates the possibility of tampering, SPU 500 may automatically destroy, or render inaccessible without privileged intervention, one or more portions of sensitive information it stores, such as execution related information and/or encryption
10 key related information. To provide further SPU operation, such destroyed information would have to be replaced by a VDE clearinghouse, administrator and/or distributor, as may be appropriate. This may be achieved by remotely downloading update and/or replacement data and/or code. In the event of a
15 disabling and/or destruction of processes and/or information as described above, the electronic appliance 600 may require a secure VDE communication with an administrator, clearinghouse, and/or distributor as appropriate in order to reinitialize the RTC 528. Some or all secure SPU 500 processes
20 may not operate until then.

It may be desirable to provide a mechanism for setting and/or synchronizing RTC 528. In the preferred embodiment, when communication occurs between VDE electronic appliance

600 and another VDE appliance, an output of RTC 528 may be compared to a controlled RTC 528 output time under control of the party authorized to be "senior" and controlling. In the event of a discrepancy, appropriate action may be taken, including
5 resetting the RTC 528 of the "junior" controlled participant in the communication.

SPU Encrypt/Decrypt Engine 522

In the preferred embodiment, SPU encrypt/decrypt engine
10 522 provides special purpose hardware (e.g., a hardware state machine) for rapidly and efficiently encrypting and/or decrypting data. In some implementations, the encrypt/decrypt functions may be performed instead by microprocessor 520 under software control, but providing special purpose encrypt/decrypt hardware
15 engine 522 will, in general, provide increased performance. Microprocessor 520 may, if desired, comprise a combination of processor circuitry and dedicated encryption/decryption logic that may be integrated together in the same circuitry layout so as to, for example, optimally share one or more circuit elements.

20

Generally, it is preferable that a computationally efficient but highly secure "bulk" encryption/decryption technique should be used to protect most of the data and objects handled by SPU 500. It is preferable that an extremely secure

encryption/decryption technique be used as an aspect of authenticating the identity of electronic appliances 600 that are establishing a communication channel and securing any transferred permission, method, and administrative information.

5 In the preferred embodiment, the encrypt/decrypt engine 522 includes both a symmetric key encryption/decryption circuit (e.g. DES, Skipjack/Clipper, IDEA, RC-2, RC-4, etc.) and an antisymmetric (asymmetric) or Public Key ("PK") encryption/decryption circuit. The public/private key
10 encryption/decryption circuit is used principally as an aspect of secure communications between an SPU 500 and VDE administrators, or other electronic appliances 600, that is between VDE secure subsystems. A symmetric encryption/decryption circuit may be used for "bulk" encrypting
15 and decrypting most data stored in secondary storage 662 of electronic appliance 600 in which SPU 500 resides. The symmetric key encryption/decryption circuit may also be used for encrypting and decrypting content stored within VDE objects 300.

20

DES or public/private key methods may be used for all encryption functions. In alternate embodiments, encryption and decryption methods other than the DES and public/private key methods could be used for the various encryption related

functions. For instance, other types of symmetric encryption/decryption techniques in which the same key is used for encryption and decryption could be used in place of DES encryption and decryption. The preferred embodiment can support a plurality of decryption/encryption techniques using multiple dedicated circuits within encrypt/decrypt engine 522 and/or the processing arrangement within SPU 500.

Pattern Matching Engine 524

Optional pattern matching engine 524 may provide special purpose hardware for performing pattern matching functions. One of the functions SPU 500 may perform is to validate/authenticate VDE objects 300 and other items. Validation/authentication often involves comparing long data strings to determine whether they compare in a predetermined way. In addition, certain forms of usage (such as logical and/or physical (contiguous) relatedness of accessed elements) may require searching potentially long strings of data for certain bit patterns or other significant pattern related metrics. Although pattern matching can be performed by SPU microprocessor 520 under software control, providing special purpose hardware pattern matching engine 524 may speed up the pattern matching process.

Compression/Decompression Engine 546

An optional compression/decompression engine 546 may be provided within an SPU 500 to, for example, compress and/or decompress content stored in, or released from, VDE objects 300.

5 Compression/decompression engine 546 may implement one or more compression algorithms using hardware circuitry to improve the performance of compression/decompression operations that would otherwise be performed by software operating on microprocessor 520, or outside SPU 500.

10 Decompression is important in the release of data such as video and audio that is usually compressed before distribution and whose decompression speed is important. In some cases, information that is useful for usage monitoring purposes (such as record separators or other delimiters) is "hidden" under a
15 compression layer that must be removed before this information can be detected and used inside SPU 500.

Random Number Generator 542

Optional random number generator 542 may provide
20 specialized hardware circuitry for generating random values (e.g., from inherently unpredictable physical processes such as quantum noise). Such random values are particularly useful for constructing encryption keys or unique identifiers, and for initializing the generation of pseudo-random sequences.

Random number generator 542 may produce values of any convenient length, including as small as a single bit per use. A random number of arbitrary size may be constructed by concatenating values produced by random number generator 542. A cryptographically strong pseudo-random sequence may be generated from a random key and seed generated with random number generator 542 and repeated encryption either with the encrypt/decrypt engine 522 or cryptographic algorithms in SPU 500. Such sequences may be used, for example, in private headers to frustrate efforts to determine an encryption key through cryptanalysis.

Arithmetic Accelerator 544

An optional arithmetic accelerator 544 may be provided within an SPU 500 in the form of hardware circuitry that can rapidly perform mathematical calculations such as multiplication and exponentiation involving large numbers. These calculations can, for example, be requested by microprocessor 520 or encrypt/decrypt engine 522, to assist in the computations required for certain asymmetric encryption/decryption operations. Such arithmetic accelerators are well-known to those skilled in the art. In some implementations, a separate arithmetic accelerator 544 may be

omitted and any necessary calculations may be performed by microprocessor 520 under software control.

DMA Controller 526

5 DMA controller 526 controls information transfers over address/data bus 536 without requiring microprocessor 520 to process each individual data transfer. Typically, microprocessor 520 may write to DMA controller 526 target and destination addresses and the number of bytes to transfer, and DMA
10 controller 526 may then automatically transfer a block of data between components of SPU 500 (e.g., from ROM 532 to RAM 534, between encrypt/decrypt engine 522 and RAM 534, between bus interface unit 530 and RAM 534, etc.). DMA controller 526 may have multiple channels to handle multiple transfers
15 simultaneously. In some implementations, a separate DMA controller 526 may be omitted, and any necessary data movements may be performed by microprocessor 520 under software control.

20 Bus Interface Unit (BIU) 530

Bus interface unit (BIU) 530 communicates information between SPU 500 and the outside world across the security barrier 502. BIU 530 shown in Figure 9 plus appropriate driver software may comprise the "appliance link" 510 shown in Figure

6. Bus interface unit 530 may be modelled after a USART or PCI bus interface in the preferred embodiment. In this example, BIU 530 connects SPU 500 to electronic appliance system bus 653 shown in Figure 8. BIU 530 is designed to prevent unauthorized access to internal components within SPU 500 and their contents. It does this by only allowing signals associated with an SPU 500 to be processed by control programs running on microprocessor 520 and not supporting direct access to the internal elements of an SPU 500.

Memory Management Unit 540

Memory Management Unit (MMU) 540, if present, provides hardware support for memory management and virtual memory management functions. It may also provide heightened security by enforcing hardware compartmentalization of the secure execution space (e.g., to prevent a less trusted task from modifying a more trusted task). More details are provided below in connection with a discussion of the architecture of a Secure Processing Environment ("SPE") 503 supported by SPU 500.

MMU 540 may also provide hardware-level support functions related to memory management such as, for example, address mapping.

SPU Memory Architecture

In the preferred embodiment, SPU 500 uses three general kinds of memory:

- (1) internal ROM 532;
- 5 (2) internal RAM 534; and
- (3) external memory (typically RAM and/or disk supplied by a host electronic appliance).

The internal ROM 532 and RAM 534 within SPU 500
10 provide a secure operating environment and execution space. Because of cost limitations, chip fabrication size, complexity and other limitations, it may not be possible to provide sufficient memory within SPU 500 to store all information that an SPU needs to process in a secure manner. Due to the practical limits
15 on the amount of ROM 532 and RAM 534 that may be included within SPU 500, SPU 500 may store information in memory external to it, and move this information into and out of its secure internal memory space on an as needed basis. In these cases, secure processing steps performed by an SPU typically
20 must be segmented into small, securely packaged elements that may be "paged in" and "paged out" of the limited available internal memory space. Memory external to an SPU 500 may not be secure. Since the external memory may not be secure, SPU 500 may encrypt and cryptographically seal code and other

information before storing it in external memory. Similarly, SPU 500 must typically decrypt code and other information obtained from external memory in encrypted form before processing (e.g., executing) based on it. In the preferred embodiment, there are two general approaches used to address potential memory limitations in a SPU 500. In the first case, the small, securely packaged elements represent information contained in secure database 610. In the second case, such elements may represent protected (e.g., encrypted) virtual memory pages. Although virtual memory pages may correspond to information elements stored in secure database 610, this is not required in this example of a SPU memory architecture.

The following is a more detailed discussion of each of these three SPU memory resources.

SPU Internal ROM

SPU 500 read only memory (ROM) 532 or comparable purpose device provides secure internal non-volatile storage for certain programs and other information. For example, ROM 532 may store "kernel" programs such as SPU control firmware 508 and, if desired, encryption key information and certain fundamental "load modules." The "kernel" programs, load module information, and encryption key information enable the

control of certain basic functions of the SPU 500. Those components that are at least in part dependent on device configuration (e.g., POST, memory allocation, and a dispatcher) may be loaded in ROM 532 along with additional load modules
5 that have been determined to be required for specific installations or applications.

In the preferred embodiment, ROM 532 may comprise a combination of a masked ROM 532a and an EEPROM and/or
10 equivalent "flash" memory 532b. EEPROM or flash memory 532b is used to store items that need to be updated and/or initialized, such as for example, certain encryption keys. An additional benefit of providing EEPROM and/or flash memory 532b is the ability to optimize any load modules and library
15 functions persistently stored within SPU 500 based on typical usage at a specific site. Although these items could also be stored in NVRAM 534b, EEPROM and/or flash memory 532b may be more cost effective.

20 Masked ROM 532a may cost less than flash and/or EEPROM 532b, and can be used to store permanent portions of SPU software/firmware. Such permanent portions may include, for example, code that interfaces to hardware elements such as the RTC 528, encryption/decryption engine 522, interrupt

handlers, key generators, etc. Some of the operating system,
library calls, libraries, and many of the core services provided by
SPU 500 may also be in masked ROM 532a. In addition, some of
the more commonly used executables are also good candidates for
5 inclusion in masked ROM 532a. Items that need to be updated
or that need to disappear when power is removed from SPU 500
should not be stored in masked ROM 532a.

Under some circumstances, RAM 534a and/or NVRAM
10 534b (NVRAM 534b may, for example, be constantly powered
conventional RAM) may perform at least part of the role of ROM
532.

SPU Internal RAM

15 SPU 500 general purpose RAM 534 provides, among other
things, secure execution space for secure processes. In the
preferred embodiment, RAM 534 is comprised of different types
of RAM such as a combination of high-speed RAM 534a and an
NVRAM ("non-volatile RAM") 534b. RAM 534a may be volatile,
20 while NVRAM 534b is preferably battery backed or otherwise
arranged so as to be non-volatile (i.e., it does not lose its contents
when power is turned off).

High-speed RAM 534a stores active code to be executed and associated data structures.

5 NVRAM 534b preferably contains certain keys and summary values that are preloaded as part of an initialization process in which SPU 500 communicates with a VDE administrator, and may also store changeable or changing information associated with the operation of SPU 500. For security reasons, certain highly sensitive information (e.g.,
10 certain load modules and certain encryption key related information such as internally generated private keys) needs to be loaded into or generated internally by SPU 500 from time to time but, once loaded or generated internally, should never leave the SPU. In this preferred embodiment, the SPU 500
15 non-volatile random access memory (NVRAM) 534b may be used for securely storing such highly sensitive information. NVRAM 534b is also used by SPU 500 to store data that may change frequently but which preferably should not be lost in a power down or power fail mode.

20

NVRAM 534b is preferably a flash memory array, but may in addition or alternatively be electrically erasable programmable read only memory (EEPROM), static RAM (SRAM), bubble memory, three dimensional holographic or other

electro-optical memory, or the like, or any other writable (e.g., randomly accessible) non-volatile memory of sufficient speed and cost-effectiveness.

5 **SPU External Memory**

The SPU 500 can store certain information on memory devices external to the SPU. If available, electronic appliance 600 memory can also be used to support any device external portions of SPU 500 software. Certain advantages may be
10 gained by allowing the SPU 500 to use external memory. As one example, memory internal to SPU 500 may be reduced in size by using non-volatile read/write memory in the host electronic appliance 600 such as a non-volatile portion of RAM 656 and/or ROM 658.

15

Such external memory may be used to store SPU programs, data and/or other information. For example, a VDE control program may be, at least in part, loaded into the memory and communicated to and decrypted within SPU 500 prior to
20 execution. Such control programs may be re-encrypted and communicated back to external memory where they may be stored for later execution by SPU 500. "Kernel" programs and/or some or all of the non-kernel "load modules" may be stored by SPU 500 in memory external to it. Since a secure database 610

may be relatively large, SPU 500 can store some or all of secure database 610 in external memory and call portions into the SPU 500 as needed.

5 As mentioned above, memory external to SPU 500 may not be secure. Therefore, when security is required, SPU 500 must encrypt secure information before writing it to external memory, and decrypt secure information read from external memory before using it. Inasmuch as the encryption layer relies on
10 secure processes and information (e.g., encryption algorithms and keys) present within SPU 500, the encryption layer effectively "extends" the SPU security barrier 502 to protect information the SPU 500 stores in memory external to it.

15 SPU 500 can use a wide variety of different types of external memory. For example, external memory may comprise electronic appliance secondary storage 652 such as a disk; external EEPROM or flash memory 658; and/or external RAM 656. External RAM 656 may comprise an external nonvolatile
20 (e.g. constantly powered) RAM and/or cache RAM.

Using external RAM local to SPU 500 can significantly improve access times to information stored externally to an SPU. For example, external RAM may be used:

- to buffer memory image pages and data structures prior to their storage in flash memory or on an external hard disk (assuming transfer to flash or hard disk can occur in significant power or system failure cases);
- 5 • provide encryption and decryption buffers for data being released from VDE objects 300.
- to cache "swap blocks" and VDE data structures currently in use as an aspect of providing a secure virtual memory environment for SPU 500.
- 10 • to cache other information in order to, for example, reduce frequency of access by an SPU to secondary storage 652 and/or for other reasons.

Dual ported external RAM can be particularly effective in improving SPU 500 performance, since it can decrease the data
15 movement overhead of the SPU bus interface unit 530 and SPU microprocessor 520.

Using external flash memory local to SPU 500 can be used to significantly improve access times to virtually all data
20 structures. Since most available flash storage devices have limited write lifetimes, flash storage needs to take into account the number of writes that will occur during the lifetime of the flash memory. Hence, flash storage of frequently written temporary items is not recommended. If external RAM is non-

volatile, then transfer to flash (or hard disk) may not be necessary.

External memory used by SPU 500 may include two categories:

- external memory dedicated to SPU 500, and
- memory shared with electronic appliance 600.

For some VDE implementations, sharing memory (e.g., electronic appliance RAM 656, ROM 658 and/or secondary storage 652) with CPU 654 or other elements of an electronic appliance 600 may be the most cost effective way to store VDE secure database management files 610 and information that needs to be stored external to SPU 500. A host system hard disk secondary memory 652 used for general purpose file storage can, for example, also be used to store VDE management files 610. SPU 500 may be given exclusive access to the external memory (e.g., over a local bus high speed connection provided by BIU 530). Both dedicated and shared external memory may be provided.

SPU Integrated Within CPU

As discussed above, it may be desirable to integrate CPU 654 and SPU 500 into the same integrated circuit and/or device.

SPU 500 shown in Figure 9 includes a microprocessor 520 that may be similar or identical to a standard microprocessor available off-the-shelf from a variety of manufacturers.

Similarly, the SPU DMA controller 526 and certain other

5 microprocessor support circuitry may be standard implementations available in off-the-shelf microprocessor and/or microcomputer chips. Since many of the general control and processing requirements provided by SPU 500 in the preferred embodiment can be satisfied using certain generic CPU and/or
10 microcontroller components, it may be desirable to integrate SPU VDE functionality into a standard generic CPU or microcontroller chip. Such an integrated solution can result in a very cost-effective "dual mode" component that is capable of performing all of the generic processing of a standard CPU as
15 well as the secure processing of an SPU. Many of the control logic functions performed by the preferred embodiment SPU can be performed by generic CPU and/or micro-controller logic so that at least a portion of the control logic does not have to be duplicated. Additional cost savings (e.g., in terms of reducing
20 manufacturing costs, inventory costs and printed circuit board real estate requirements) may also be obtained by not requiring an additional, separate physical SPU 500 device or package.

Figure 9A shows one example architecture of a combination CPU/SPU 2650. CPU/SPU 2650 may include a standard

microprocessor or microcontroller 2652, a standard bus interface unit (BIU) 2656, and a standard (optional) DMA controller 2654, as well as various other standard I/O controllers, computation circuitry, etc. as may be found in a typical off-the-shelf microprocessor/microcontroller. Real time clock 528 may be added to the standard architecture to give the CPU/SPU 2650 access to the real time clock functions as discussed above in connection with Figure 9. Real-time clock 528 must be protected from tampering in order to be secure. Such protections may include internal or external backup power, an indication that its power (and thus its operation) has been interrupted, and/or an indication that the external clock signal(s) from which it derives its timing have been interfered with (e.g., sped up, slowed down). Similarly, an encrypt/decrypt engine 522, pattern matching engine 524, compression/decompression engine 546 and/or arithmetic accelerator 544 may be added if desired to provide greater efficiencies, or the functions performed by these components could be provided instead by software executing on microprocessor 2652. An optional memory management unit 540 may also be provided if desired. A true random number generator 542 may be provided also if desired. Connections shown between mode interface switch 2658 and other components can carry both data and control information, specifically control information that determines what security-

relevant aspects of the other components are available for access and/or manipulation.

5 In addition, secure ROM 532 and/or secure RAM 534 may be provided within CPU/SPU 2650 along with a "mode interface switch" 2658a, 2658b. Mode interface switch 2658 selectively provides microprocessor 2652 with access to secure memory 532, 534 and other secure components (blocks 522, 546, 524, 542, 544, 528) depending upon the "mode" CPU/SPU 2650 is operating in.
10 CPU/SPU 2650 in this example may operate in two different modes:

- an "SPU" mode, or
- a "normal" mode.

In the "normal" mode, CPU/SPU 2650 operates
15 substantially identically to a standard off-the-shelf CPU while also protecting the security of the content, state, and operations of security-relevant components included in CPU/SPU 2650. Such security-relevant components may include the secure memories 532, 534; the encrypt/decrypt engine 522, the optional
20 pattern-matching engine 524, random number generator 542, arithmetic accelerator 544, the SPU-not-initialized flag 2671, the secure mode interface switch 2658, the real-time clock 528, the DMA controller 2654, the MMU 540, compress/decompress block

546, and/or any other components that may affect security of the operation of the CPU/SPU in "SPU" mode.

5 In this example, CPU/SPU 2650 operating in the "normal" mode controls mode interface switch 2658 to effectively "disconnect" (i.e., block unsecure access to) the security-relevant components, or to the security-relevant aspects of the operations of such components as have a function for both "normal" and "SPU" mode. In the "normal" mode, for example, microprocessor 10 2652 could access information from standard registers or other internal RAM and/or ROM (not shown), execute instructions in a "normal" way, and perform any other tasks as are provided within a standard CPU -- but could not access or compromise the contents of secure memory 532, 534 or access blocks 522, 524, 15 542, 544, 546. In this example "normal" mode, mode interface switch 2658 would effectively prevent any access (e.g., both read and write access) to secure memory 532, 534 so as to prevent the information stored within that secure memory from being compromised.

20 When CPU/SPU 2650 operates in the "SPU" mode, mode interface switch 2658 allows microprocessor 2652 to access secure memory 532, 534, and to control security-relevant aspects of other components in the CPU/SPU. The "SPU" mode in this

example requires all instructions executed by microprocessor 2652 to be fetched from secure memory 532, 534 -- preventing execution based on "mixed" secure and non-secure instructions. In the "SPU" mode, mode interface switch 2658 may, in one example embodiment, disconnect or otherwise block external accesses carried over bus 652 from outside CPU/SPU 2650 (e.g., DMA accesses, cache coherency control accesses) to ensure that the microprocessor 2652 is controlled entirely by instructions carried within or derived from the secure memory 532, 534. Mode interface switch 2658 may also disconnect or otherwise block access by microprocessor 2652 to some external memory and/or other functions carried over bus 652. Mode interface switch 2658 in this example prevents other CPU operations/instructions from exposing the contents of secure memory 532, 534.

In the example shown in Figure 9A, the mode control of mode interface switch 2658 is based on a "mode" control signal provided by microprocessor 2652. In this example, microprocessor 2652 may be slightly modified so it can execute two "new" instructions:

- "enable 'SPU' mode" instruction, and
- "disable 'SPU' mode" instruction.

When microprocessor 2652 executes the "enable 'SPU' mode" instruction, it sends an appropriate "mode" control signal to mode interface switch 2658 to "switch" the interface switch into the "SPU" mode of operation. When microprocessor 2652 executes the "disable 'SPU' mode" instruction, it sends an appropriate "mode" control signal to mode interface switch 2658 to disable the "SPU" mode of operation.

When CPU/SPU 2650 begins operating in the "SPU" mode (based on microprocessor 2652 executing the "enable "SPU" mode" instruction), mode interface switch 2658 forces microprocessor 2652 to begin fetching instructions from secure memory 532, 534 (e.g., beginning at some fixed address) in one example. When CPU/SPU 2650 begins operating in this example "SPU" mode, mode interface switch 2658 may force microprocessor 2652 to load its registers from some fixed address in secure memory 532, 534 and may begin execution based on such register content. Once operating in the "SPU" mode, microprocessor 2652 may provide encryption/decryption and other control capabilities based upon the code and other content of secure memory 532, 534 needed to provide the VDE

functionality of SPU 500 described above. For example,
microprocessor 2652 operating under control of information
within secure memory 532, 534 may read encrypted information
from bus 652 via bus interface unit 2656, write decrypted
5 information to the bus interface unit, and meter and limit
decryption of such information based on values stored in the
secure memory.

At the end of secure processing, execution by
10 microprocessor 2652 of the "disable SPU mode" instruction may
cause the contents of all registers and other temporary storage
locations used by microprocessor 2652 that are not within secure
memory 532, 534 to be destroyed or copied into secure memory
532, 534 before "opening" mode interface switch 2658. Once
15 mode interface switch 2658 is "open," the microprocessor 2652 no
longer has access to secure memory 532, 534 or the information
it contained, or to control or modify the state of any other
security-relevant components or functions contained within
CPU/SPU 2650 to which access is controlled by mode interface
20 switch 2658.

Whenever CPU/SPU 2650 enters or leaves the "SPU"
mode, the transition is performed in such a way that no
information contained in the secure memory 532, 534 or derived

from it (e.g., stored in registers or a cache memory associated with microprocessor 2652) while in the "SPU" mode can be exposed by microprocessor 2652 operations that occur in the "normal" mode. This may be accomplished either by hardware mechanisms that protect against such exposure, software instructions executed in "SPU" mode that clear, reinitialize, and otherwise reset during such transitions, or a combination of both.

In some example implementations, interrupts may be enabled while CPU/SPU 2650 is operating in the "SPU" mode similarly interrupts and returns from interrupts while in the "SPU" mode may allow transitions from "SPU" mode to "normal" mode and back to "SPU" mode without exposing the content of secure memory 532, 534 or the content of registers or other memory associated with microprocessor 2652 that may contain information derived from secure mode operation.

In some example implementations, there may be CPU/SPU activities such as DMA transfers between external memory and/or devices and secure memory 532, 534 that are initiated by microprocessor 2652 but involve autonomous activity by DMA controller 2654 and, optionally, encrypt/decrypt engine 522 and/or compress/decompress engine 546. In such implementations, mode interface switch 2658 and its associated

control signals may be configured to permit such pending activities (e.g. DMA transfers) to continue to completion even after CPU/SPU 2650 leaves "SPU" mode, provided that upon completion, all required clearing, reinitialization, and/or reset activities occur, and provided that no access or interference is permitted with the pending activities except when CPU/SPU 2650 is operating in "SPU" mode.

In an additional example embodiment, encryption/decryption logic may be connected between microprocessor 2652 and secure memory 532, 534. This additional encryption/decryption logic may be connected "in parallel" to mode interface switch 2658. The additional encryption/decryption logic may allow certain accesses by microprocessor 2652 to the secure memory 532, 534 when CPU/SPU 2650 is operating in the "normal" mode. In this alternate embodiment, reads from secure memory 532, 534 when CPU/SPU 2650 is operating in the "normal" mode automatically result in the read information being encrypted before it is delivered to microprocessor 2652 (and similarly, and writes to the secure memory may result in the written information being decrypted before it is deposited into the secure memory). This alternative embodiment may permit access to secure memory 532, 534 (which may in this example store the information in

"clear" form) by microprocessor 2652 when CPU/SPU 2650 is operating in the "non-secure normal" mode, but only reveals the secure memory contents to microprocessor 2652 in unencrypted form when the CPU/SPU is operating in the "SPU" mode. Such access may also be protected by cryptographic authentication techniques (e.g., message authentication codes) to prevent modification or replay attacks that modify encrypted data stored in secure memory 532, 534. Such protection may be performed utilizing either or both of software and/or hardware cryptographic techniques.

All of the components shown in Figure 9A may be disposed within a single integrated circuit package. Alternatively, mode interface switch 2658 and secure memory 532, 534, and other security-relevant components might be placed within an integrated circuit chip package and/or other package separate from the rest of CPU/SPU 2650. In this two-package version, a private bus could be used to connect microprocessor 2652 to the mode interface switch 2658 and associated secure memory 532, 534. To maintain security in such multi-package versions, it may be necessary to enclose all the packages and their interconnections in an external physical tamper-resistant barrier.

Initialization of Integrated CPU/SPU

Instructions and/or data may need to be loaded into CPU/SPU 2650 before it can operate effectively as an SPU 500.

5 This may occur during the manufacture of CPU/SPU 2650 or subsequently at a CPU/SPU initialization facility. Security of such initialization may depend on physical control of access to the CPU/SPU component(s), on cryptographic means, or on some combination of both. Secure initialization may be performed in plural steps under the control of different parties, such that an
10 initialization step to be performed by party B is preconditioned on successful performance of a step by party A. Different initialization steps may be protected using different security techniques (e.g. physical access, cryptography).

15

In this example, switch 2658 may expose an external control signal 2670 that requests operation in "SPU" mode rather than "normal" mode after a power-on reset. This signal would be combined (e.g., by a logical AND 2672) with a non-volatile
20 storage element 2671 internal to CPU/SPU 2650. If both of these signals are asserted, AND gate 2672 would cause CPU/SPU 2650 to begin operating in SPU mode, either executing existing instructions from an address in SPU memory 532, executing instructions from main memory 2665 or otherwise external to the

CPU/SPU. The instructions thus executed would permit arbitrary initialization and other functions to be performed in "SPU" mode without necessarily requiring any instructions to be previously resident in the SPU memory 532.

5

Once initialized, the SPU would, under control of its initialization program, indicate to switch 2658 that the flag 2671 is to be cleared. Clearing flag 2671 would permanently disable this initialization capability because no mechanism would be provided to set flag 2671 back to its initial value.

10

If flag 2671 is clear, or control signal 2670 is not asserted, CPU/SPU 2650 would behave precisely as does microprocessor 2652 with respect to power-on reset and other external conditions. Under such conditions, only execution of the "enable SPU mode" instruction or otherwise requesting SPU mode under

15

program control would cause "SPU" mode to be entered.

Additionally, a mechanism could be provided to permit microprocessor 2652 and/or control signal 2672 to reinitialize the flag 2671. Such reinitialization would be performed in a manner that cleared secure memory 532, 534 of any security-relevant information and reinitialized the state of all security-relevant components. This reinitialization mechanism would permit CPU/SPU 2650 to be initialized several times, facilitating testing

20

and/or re-use for different applications, while protecting all security-relevant aspects of its operation.

5 In the preferred embodiment, CPU/SPU 2650 would, when SPU mode has not yet been established, begin operating in SPU mode by fetching instructions from secure non-volatile memory 532, thereby ensuring a consistent initialization sequence and preventing SPU dependence on any information held outside CPU/SPU 2650. This approach permits secret initialization
10 information (e.g., keys for validating digital signatures on additional information to be loaded into secure memory 532, 534) to be held internally to CPU/SPU 2650 so that it is never exposed to outside access. Such information could even be supplied by a hardware "mask" used in the semiconductor fabrication process.

15

CPU/SPU Integrated With Unmodified Microprocessor

Figure 9B shows an additional example embodiment, in which a completely standard microprocessor 2652 integrated circuit chip could be transformed into a CPU/SPU 2650 by
20 adding an SPU chip 2660 that mediates access to external I/O devices and memory. In such an embodiment, the microprocessor 2652 would be connected to the SPU chip 2660 by a private memory bus 2661, and all three such components

would be contained within hardware tamper-resistant barrier
502.

5 In this embodiment, SPU chip 2660 may have the same
secure components as in Figure 9, i.e., it may have a
ROM/EEPROM 532, a RAM 532, an RTC 528, an (optional)
encryption/decryption engine 522, an (optional) random number
generator (RNG) 542, an (optional) arithmetic accelerator 544,
and a (optional) compression/decompression engine 546, and a
10 (optional) pattern matching circuit 524. Microprocessor 520 is
omitted from SPU chip 2660 since the standard microprocessor
2650 performs the processing functions instead. In addition,
SPU chip 2660 may include a flag 2671 and AND gate logic 2672
for the initialization purposes discussed above.

15

In addition, SPU chip 2660 includes an enhanced switch
2663 that provides the same overall (bus enhanced) functionality
performed by the switch 2658 in the Figure 9A embodiment.

20

Enhanced switch 2663 would perform the functions of a
bus repeater, mediator and interpreter. For example, enhanced
switch 2663 may act as a bus repeater that enables
microprocessor 2652's memory accesses made over internal
memory bus 2661 to be reflected to external memory bus 2664

and performed on main memory 2665. Enhanced switch 2663 may also act as a bus repeater similarly for internal I/O bus 2662 to external I/O bus 2665 in the event that microprocessor 2652 performs I/O operations distinctly from memory operations.

5 Enhanced switch 2663 may also perform the function of a mediator for microprocessor control functions 2666 (e.g., non-maskable interrupt, reset) with respect to externally requested control functions 2667. Enhanced switch 2663 may also provide mediation for access to SPU-protected resources
10 such as ROM 532, RAM 534, encrypt/decrypt engine 522 (if present), random number generator 542 (if present), arithmetic accelerator 544 (if present), pattern matching engine 524 (if present), and real-time clock 528 (if present). Enhanced switch 2663 may also act as an interpreter of control signals received
15 from microprocessor 2652 indicating entry to, exit from, and control of SPU mode.

Switch 2663 in this example recognizes a specific indication (e.g., an instruction fetch access to a designated
20 address in the secure memory 532) as the equivalent to the "enable 'SPU' mode" instruction. Upon recognizing such an indication, it may isolate the CPU/SPU 2650 from external buses and interfaces 2664, 2665, and 2667 such that any external activity, such as DMA cycles, would be "held" until the switch

2663 permits access again. After this, switch 2663 permits a single access to a specific location in secure memory 532 to complete.

5 The single instruction fetched from the designated location performs a control operation (a cache flush, for example), that can only be performed in microprocessor 2652's most privileged operating mode, and that has an effect visible to switch 2663. Switch 2663 awaits the occurrence of this event, and if it does
10 not occur within the expected number of cycles, does not enter "SPU" mode.

 Occurrence of the control operation demonstrates that microprocessor 2652 is executing in its most privileged "normal"
15 mode and therefore can be trusted to execute successfully the "enter 'SPU' mode" sequence of instructions stored in secure memory 532. If microprocessor 2652 were not executing in its most privileged mode, there would be no assurance that those instructions would execute successfully. Because switch 2663
20 isolates microprocessor 2652 from external signals (e.g., interrupts) until "SPU" mode is successfully initialized, the entry instructions can be guaranteed to complete successfully.

Following the initial instruction, switch 2663 can enter "partial SPU mode," in which a restricted area of ROM 532 and RAM 534 may be accessible. Subsequent instructions in secure memory 532 may then be executed by microprocessor 2652 to place it into a known state such that it can perform SPU functions -- saving any previous state in the restricted area of RAM 534 that is accessible. After the known state is established, an instruction may be executed to deliver a further indication (e.g., a reference to another designated memory location) to switch 2663, which would enter "SPU" mode. If this further indication is not received within the expected interval, switch 2663 will not enter "SPU" mode. Once in "SPU" mode, switch 2663 permits access to all of ROM 532, RAM 534, and other devices in SPU chip 2660.

15

The instructions executed during "partial SPU" mode must be carefully selected to ensure that no similar combination of instructions and processor state could result in a control transfer out of the protected SPU code in ROM 532 or RAM 534. For example, internal debugging features of microprocessor 2652 must be disabled to ensure that a malicious program could not set up a breakpoint later within protected SPU code and receive control. Similarly, all address translation must be disabled or reinitialized to ensure that previously created MMU data

20

structures would not permit SPU memory accesses to be compromised. The requirement that the instructions for "partial SPU mode" run in the microprocessor 2652's most privileged mode is necessary to ensure that all its processor control
5 functions can be effectively disabled.

The switch 2663 provides additional protection against tampering by ensuring that the expected control signals occur after an appropriate number of clock cycles. Because the "partial
10 SPU" initialization sequence is entirely deterministic, it is not feasible for malicious software to interfere with it and still retain the same timing characteristics, even if malicious software is running in microprocessor 2652's most privileged mode.

15 Once in "SPU" mode, switch 2663 may respond to additional indications or signals generated by microprocessor 2652 (e.g., references to specific memory addresses) controlling features of SPU mode. These might include enabling access to external buses 2664 and 2665 so that SPU-protected code could
20 reference external memory or devices. Any attempts by components outside CPU/SPU 2650 to perform operations (e.g., accesses to memory, interrupts, or other control functions) may be prevented by switch 2663 unless they had been explicitly enabled by instructions executed after "SPU" mode is entered.

To leave SPU mode and return to normal operation, the instructions executing in "SPU" mode may provide a specific indication to switch 2663 (e.g., a transfer to a designated memory address). This indication may be recognized by switch 2663 as
5 indicating a return to "normal mode," and it may again restrict access to ROM 532, RAM 534, and all other devices within SPU chip 2660, while re-enabling external buses and control lines 2664, 2665, and 2667. The instructions executed subsequently may restore the CPU state to that which was saved on entry to
10 SPU mode, so that microprocessor 2652 may continue to perform functions in progress when the SPU was invoked.

In an alternate embodiment, the entry into SPU mode may be conditioned on an indication recognized by switch 2663, but
15 the switch may then use a hardware mechanism (e.g., the processor's RESET signal) to reinitialize microprocessor 2562. In such an embodiment, switch 2663 may not implement partial SPU mode, but may instead enter SPU mode directly and ensure that the address from which instructions would be fetched by
20 microprocessor 2652 (specific to microprocessor 2652's architecture) results in accesses to appropriate locations in the SPU memory 532. This could reduce the complexity of the SPU mode entry mechanisms in switch 2663, but could incur an

additional processing cost from using a different reinitialization mechanism for microprocessor 2652.

5 SPU chip 2660 may be customized to operate in conjunction with a particular commercial microprocessor. In this example, the SPU may be customized to contain at least the specialized "enter SPU mode" instruction sequences to reinitialize the processor's state and, to recognize special indications for SPU control operations. SPU chip 2660 may also
10 be made electrically compatible with microprocessor 2652's external bus interfaces. This compatibility would permit CPU/SPU 2650 to be substituted for microprocessor 2652 without change either to software or hardware elsewhere in a computer system.

15 In other alternate embodiments, the functions described above for SPU chip 2600, microprocessor 2652, and internal buses 2661, 2662, and 2666 could all be combined within a single integrated circuit package, and/or on a single silicon die. This
20 could reduce packaging complexity and/or simplify establishment of the hardware tamper-resistant barrier 502.

* * * * *

The hardware configuration of an example of electronic appliance 600 has been described above. The following section describes an example of the software architecture of electronic appliance 600 provided by the preferred embodiment, including the structure and operation of preferred embodiment "Rights Operating System" ("ROS") 602.

Rights Operating System 602

Rights Operating System ("ROS") 602 in the preferred embodiment is a compact, secure, event-driven, services-based, "component" oriented, distributed multiprocessing operating system environment that integrates VDE information security control information, components and protocols with traditional operating system concepts. Like traditional operating systems, ROS 602 provided by the preferred embodiment is a piece of software that manages hardware resources of a computer system and extends management functions to input and/or output devices, including communications devices. Also like traditional operating systems, preferred embodiment ROS 602 provides a coherent set of basic functions and abstraction layers for hiding the differences between, and many of the detailed complexities of, particular hardware implementations. In addition to these characteristics found in many or most operating systems, ROS 602 provides secure VDE transaction management and other

advantageous features not found in other operating systems.

The following is a non-exhaustive list of some of the advantageous features provided by ROS 602 in the preferred embodiment:

5

Standardized interface provides coherent set of basic functions

- simplifies programming
- the same application can run on many different platforms

Event driven

10

- eases functional decomposition
- extendible
- accommodates state transition and/or process oriented events

- simplifies task management

15

- simplifies inter-process communications

Services based

- allows simplified and transparent scalability
- simplifies multiprocessor support
- hides machine dependencies

20

- eases network management and support

Component Based Architecture

- processing based on independently deliverable secure components

- component model of processing control allows different sequential steps that are reconfigurable based on requirements
 - components can be added, deleted or modified (subject to permissioning)
 - full control information over pre-defined and user-defined application events
 - events can be individually controlled with independent executables
- 10 Secure
- secure communications
 - secure control functions
 - secure virtual memory management
 - information control structures protected from exposure
 - data elements are validated, correlated and access controlled
 - components are encrypted and validated independently
 - components are tightly correlated to prevent unauthorized use of elements
- 20 • control structures and secured executables are validated prior to use to protect against tampering
- integrates security considerations at the I/O level
 - provides on-the-fly decryption of information at release time

- enables a secure commercial transaction network
- flexible key management features

Scalaeble

- highly scalaeble across many different platforms
- 5 • supports concurrent processing in a multiprocessor environment
- supports multiple cooperating processors
- any number of host or security processors can be supported
- control structures and kernel are easily portable to various
- 10 host platforms and to different processors within a target platform without recompilation
- supports remote processing
- Remote Procedure Calls may be used for internal OS communications

15 Highly Integratable

- can be highly integrated with host platforms as an additional operating system layer
- permits non-secure storage of secured components and information using an OS layer "on top of" traditional OS
- 20 platforms
- can be seamlessly integrated with a host operating system to provide a common usage paradigm for transaction management and content access

- integration may take many forms: operating system layers for desktops (e.g., DOS, Windows, Macintosh); device drivers and operating system interfaces for network services (e.g., Unix and Netware); and dedicated component drivers for "low end" set tops are a few of many examples
- can be integrated in traditional and real time operating systems

Distributed

- provides distribution of control information and reciprocal control information and mechanisms
- supports conditional execution of controlled processes within any VDE node in a distributed, asynchronous arrangement
- controlled delegation of rights in a distributed environment
- supports chains of handling and control
- management environment for distributed, occasionally connected but otherwise asynchronous networked database
- real time and time independent data management
- supports "agent" processes

Transparent

- can be seamlessly integrated into existing operating systems

- can support applications not specifically written to use it

Network friendly

- internal OS structures may use RPCs to distribute processing
- 5 • subnets may seamlessly operate as a single node or independently

General Background Regarding Operating Systems

10 An "operating system" provides a control mechanism for organizing computer system resources that allows programmers to create applications for computer systems more easily. An operating system does this by providing commonly used functions, and by helping to ensure compatibility between different computer hardware and architectures (which may, for
15 example, be manufactured by different vendors). Operating systems also enable computer "peripheral device" manufacturers to far more easily supply compatible equipment to computer manufacturers and users.

20 Computer systems are usually made up of several different hardware components. These hardware components include, for example:

a central processing unit (CPU) for executing instructions;

an array of main memory cells (e.g., "RAM" or "ROM") for storing instructions for execution and data acted upon or parameterizing those instructions; and

5 one or more secondary storage devices (e.g., hard disk drive, floppy disk drive, CD-ROM drive, tape reader, card reader, or "flash" memory) organized to reflect named elements (a "file system") for storing images of main memory cells.

10 Most computer systems also include input/output devices such as keyboards, mice, video systems, printers, scanners and communications devices.

To organize the CPU's execution capabilities with
15 available RAM, ROM and secondary storage devices, and to provide commonly used functions for use by programmers, a piece of software called an "operating system" is usually included with the other components. Typically, this piece of software is designed to begin executing after power is applied to the
20 computer system and hardware diagnostics are completed.

Thereafter, all use of the CPU, main memory and secondary memory devices is normally managed by this "operating system" software. Most computer operating systems also typically include a mechanism for extending their management functions

to I/O and other peripheral devices, including commonly used functions associated with these devices.

5 By managing the CPU, memory and peripheral devices through the operating system, a coherent set of basic functions and abstraction layers for hiding hardware details allows programmers to more easily create sophisticated applications. In addition, managing the computer's hardware resources with an operating system allows many differences in design and
10 equipment requirements between different manufacturers to be hidden. Furthermore, applications can be more easily shared with other computer users who have the same operating system, with significantly less work to support different manufacturers' base hardware and peripheral devices.

15

ROS 602 is an Operating System Providing Significant Advantages

ROS 602 is an "operating system." It manages the resources of electronic appliance 600, and provides a commonly
20 used set of functions for programmers writing applications 608 for the electronic appliance. ROS 602 in the preferred embodiment manages the hardware (e.g., CPU(s), memory(ies), secure RTC(s), and encrypt/decrypt engines) within SPU 500. ROS may also manage the hardware (e.g., CPU(s) and

memory(ies)) within one or more general purpose processors within electronic appliance 600. ROS 602 also manages other electronic appliance hardware resources, such as peripheral devices attached to an electronic appliance. For example, referring to Figure 7, ROS 602 may manage keyboard 612, display 614, modem 618, disk drive 620, printer 622, scanner 624. ROS 602 may also manage secure database 610 and a storage device (e.g., "secondary storage" 652) used to store secure database 610.

ROS 602 supports multiple processors. ROS 602 in the preferred embodiment supports any number of local and/or remote processors. Supported processors may include at least two types: one or more electronic appliance processors 654, and/or one or more SPUs 500. A host processor CPU 654 may provide storage, database, and communications services. SPU 500 may provide cryptographic and secured process execution services. Diverse control and execution structures supported by ROS 602 may require that processing of control information occur within a controllable execution space -- this controllable execution space may be provided by SPU 500. Additional host and/or SPU processors may increase efficiencies and/or capabilities. ROS 602 may access, coordinate and/or manage further processors remote to an electronic appliance 600 (e.g., via

network or other communications link) to provide additional processor resources and/or capabilities.

5 ROS 602 is services based. The ROS services provided using a host processor 654 and/or a secure processor (SPU 500) are linked in the preferred embodiment using a "Remote Procedure Call" ("RPC") internal processing request structure. Cooperating processors may request interprocess services using a RPC mechanism, which is minimally time dependent and can be distributed over cooperating processors on a network of hosts. 10 The multi-processor architecture provided by ROS 602 is easily extensible to support any number of host or security processors. This extensibility supports high levels of scalability. Services also allow functions to be implemented differently on different 15 equipment. For example, a small appliance that typically has low levels of usage by one user may implement a database service using very different techniques than a very large appliance with high levels of usage by many users. This is another aspect of scalability.

20

ROS 602 provides a distributed processing environment. For example, it permits information and control structures to automatically, securely pass between sites as required to fulfill a user's requests. Communications between VDE nodes under the

distributed processing features of ROS 602 may include
interprocess service requests as discussed above. ROS 602
supports conditional and/or state dependent execution of
controlled processors within any VDE node. The location that
5 the process executes and the control structures used may be
locally resident, remotely accessible, or carried along by the
process to support execution on a remote system.

ROS 602 provides distribution of control information,
10 including for example the distribution of control structures
required to permit "agents" to operate in remote environments.
Thus, ROS 602 provides facilities for passing execution and/or
information control as part of emerging requirements for "agent"
processes.

15 If desired, ROS 602 may independently distribute control
information over very low bandwidth connections that may or
may not be "real time" connections. ROS 602 provided by the
preferred embodiment is "network friendly," and can be
20 implemented with any level of networking protocol. Some
examples include e-mail and direct connection at approximately
"Layer 5" of the ISO model.

The ROS 602 distribution process (and the associated auditing of distributed information) is a controlled event that itself uses such control structures. This "reflective" distributed processing mechanism permits ROS 602 to securely distribute rights and permissions in a controlled manner, and effectively restrict the characteristics of use of information content. The controlled delegation of rights in a distributed environment and the secure processing techniques used by ROS 602 to support this approach provide significant advantages.

Certain control mechanisms within ROS 602 are "reciprocal." Reciprocal control mechanisms place one or more control components at one or more locations that interact with one or more components at the same or other locations in a controlled way. For example, a usage control associated with object content at a user's location may have a reciprocal control at a distributor's location that governs distribution of the usage control, auditing of the usage control, and logic to process user requests associated with the usage control. A usage control at a user's location (in addition to controlling one or more aspects of usage) may prepare audits for a distributor and format requests associated with the usage control for processing by a distributor. Processes at either end of a reciprocal control may be further controlled by other processes (e.g., a distributor may be limited

by a budget for the number of usage control mechanisms they may produce). Reciprocal control mechanisms may extend over many sites and many levels (e.g., a creator to a distributor to a user) and may take any relationship into account (e.g., creator/distributor, distributor/user, user/user, user/creator, user/creator/distributor, etc.) Reciprocal control mechanisms have many uses in VDE 100 in representing relationships and agreements in a distributed environment.

10 ROS 602 is scalable. Many portions of ROS 602 control structures and kernel(s) are easily portable to various host platforms without recompilation. Any control structure may be distributed (or redistributed) if a granting authority permits this type of activity. The executable references within ROS 602 are
15 portable within a target platform. Different instances of ROS 602 may execute the references using different resources. For example, one instance of ROS 602 may perform a task using an SPU 500, while another instance of ROS 602 might perform the same task using a host processing environment running in
20 protected memory that is emulating an SPU in software. ROS 602 control information is similarly portable; in many cases the event processing structures may be passed between machines and host platforms as easily as between cooperative processors in a single computer. Appliances with different levels of usage

and/or resources available for ROS 602 functions may implement those functions in very different ways. Some services may be omitted entirely if insufficient resources exist. As described elsewhere, ROS 602 "knows" what services are available, and
5 how to proceed based on any given event. Not all events may be processable if resources are missing or inadequate.

ROS 602 is component based. Much of the functionality provided by ROS 602 in the preferred embodiment may be based
10 on "components" that can be securely, independently deliverable, replaceable and capable of being modified (e.g., under appropriately secure conditions and authorizations). Moreover, the "components" may themselves be made of independently deliverable elements. ROS 602 may assemble these elements
15 together (using a construct provided by the preferred embodiment called a "channel") at execution time. For example, a "load module" for execution by SPU 500 may reference one or more "method cores," method parameters and other associated data structures that ROS 602 may collect and assemble together
20 to perform a task such as billing or metering. Different users may have different combinations of elements, and some of the elements may be customizable by users with appropriate authorization. This increases flexibility, allows elements to be reused, and has other advantages.

ROS 602 is highly secure. ROS 602 provides mechanisms to protect information control structures from exposure by end users and conduit hosts. ROS 602 can protect information, VDE control structures and control executables using strong encryption and validation mechanisms. These encryption and validation mechanisms are designed to make them highly resistant to undetected tampering. ROS 602 encrypts information stored on secondary storage device(s) 652 to inhibit tampering. ROS 602 also separately encrypts and validates its various components. ROS 602 correlates control and data structure components to prevent unauthorized use of elements. These features permit ROS 602 to independently distribute elements, and also allows integration of VDE functions 604 with non-secure "other" OS functions 606.

15

ROS 602 provided by the preferred embodiment extends conventional capabilities such as, for example, Access Control List (ACL) structures, to user and process defined events, including state transitions. ROS 602 may provide full control information over pre-defined and user-defined application events. These control mechanisms include "go/no-go" permissions, and also include optional event-specific executables that permit complete flexibility in the processing and/or controlling of events. This structure permits events to be

20

individually controlled so that, for example, metering and budgeting may be provided using independent executables. For example, ROS 602 extends ACL structures to control arbitrary granularity of information. Traditional operating systems provide static "go-no go" control mechanisms at a file or resource level; ROS 602 extends the control concept in a general way from the largest to the smallest sub-element using a flexible control structure. ROS 602 can, for example, control the printing of a single paragraph out of a document file.

ROS 602 provided by the preferred embodiment permits secure modification and update of control information governing each component. The control information may be provided in a template format such as method options to an end-user. An end-user may then customize the actual control information used within guidelines provided by a distributor or content creator. Modification and update of existing control structures is preferably also a controllable event subject to auditing and control information.

ROS 602 provided by the preferred embodiment validates control structures and secured executables prior to use. This validation provides assurance that control structures and executables have not been tampered with by end-users. The

validation also permits ROS 602 to securely implement components that include fragments of files and other operating system structures. ROS 602 provided by the preferred embodiment integrates security considerations at the operating system I/O level (which is below the access level), and provides "on-the-fly" decryption of information at release time. These features permit non-secure storage of ROS 602 secured components and information using an OS layer "on top of" traditional operating system platforms.

ROS 602 is highly integratable with host platforms as an additional operating system layer. Thus, ROS 602 may be created by "adding on" to existing operating systems. This involves hooking VDE "add ons" to the host operating system at the device driver and network interface levels. Alternatively, ROS 602 may comprise a wholly new operating system that integrates both VDE functions and other operating system functions.

Indeed, there are at least three general approaches to integrating VDE functions into a new operating system, potentially based on an existing operating system, to create a Rights Operating System 602 including:

- (1) Redesign the operating system based on VDE transaction management requirements;
- (2) Compile VDE API functions into an existing operating systems; and
- 5 (3) Integrate a VDE Interpreter into an existing operating system.

The first approach could be most effectively applied when a new operating system is being designed, or if a significant
10 upgrade to an existing operating system is planned. The transaction management and security requirements provided by the VDE functions could be added to the design requirements list for the design of a new operating system that provides, in an optimally efficient manner, an integration of "traditional"
15 operating system capabilities and VDE capabilities. For example, the engineers responsible for the design of the new version or instance of an operating system would include the requirements of VDE metering/transaction management in addition to other requirements (if any) that they use to form their design
20 approach, specifications, and actual implementations. This approach could lead to a "seamless" integration of VDE functions and capabilities by threading metering/transaction management functionality throughout the system design and implementation.

The second approach would involve taking an existing set of API (Application Programmer Interface) functions, and incorporating references in the operating system code to VDE function calls. This is similar to the way that the current
5 Windows operating system is integrated with DOS, wherein DOS serves as both the launch point and as a significant portion of the kernel underpinning of the Windows operating system. This approach would be also provide a high degree of "seamless" integration (although not quite as "seamless" as the first
10 approach). The benefits of this approach include the possibility that the incorporation of metering/transaction management functionality into the new version or instance of an operating system may be accomplished with lower cost (by making use of the existing code embodied in an API, and also using the design
15 implications of the API functional approach to influence the design of the elements into which the metering/transaction management functionality is incorporated).

The third approach is distinct from the first two in that it
20 does not incorporate VDE functionality associated with metering/transaction management and data security directly into the operating system code, but instead adds a new generalized capability to the operating system for executing metering/transaction management functionality. In this case, an

interpreter including metering/transaction management functions would be integrated with other operating system code in a "stand alone" mode. This interpreter might take scripts or other inputs to determine what metering/transaction management functions should be performed, and in what order and under which circumstances or conditions they should be performed.

Instead of (or in addition to) integrating VDE functions into/with an electronic appliance operating system, it would be possible to provide certain VDE functionality available as an application running on a conventional operating system.

ROS Software Architecture

Figure 10 is a block diagram of one example of a software structure/architecture for Rights Operating System ("ROS") 602 provided by the preferred embodiment. In this example, ROS 602 includes an operating system ("OS") "core" 679, a user Application Program Interface ("API") 682, a "redirector" 684, an "intercept" 692, a User Notification/Exception Interface 686, and a file system 687. ROS 602 in this example also includes one or more Host Event Processing Environments ("HPEs") 655 and/or one or more Secure Event Processing Environments ("SPEs") 503

(these environments may be generically referred to as "Protected Processing Environments" 650).

5 HPE(s) 655 and SPE(s) 503 are self-contained computing and processing environments that may include their own operating system kernel 688 including code and data processing resources. A given electronic appliance 600 may include any number of SPE(s) 503 and/or any number of HPE(s) 655. HPE(s) 655 and SPE(s) 503 may process information in a secure way,
10 and provide secure processing support for ROS 602. For example, they may each perform secure processing based on one or more VDE component assemblies 690, and they may each offer secure processing services to OS kernel 680.

15 In the preferred embodiment, SPE 503 is a secure processing environment provided at least in part by an SPU 500. Thus, SPU 500 provides the hardware tamper-resistant barrier 503 surrounding SPE 503. SPE 503 provided by the preferred embodiment is preferably:

- 20
- small and compact
 - loadable into resource constrained environments such as for example minimally configured SPUs 500
 - dynamically updatable

- extensible by authorized users
- integratable into object or procedural environments
- secure.

5

In the preferred embodiment, HPE 655 is a secure processing environment supported by a processor other than an SPU, such as for example an electronic appliance CPU 654 general-purpose microprocessor or other processing system or device. In the preferred embodiment, HPE 655 may be considered to "emulate" an SPU 500 in the sense that it may use software to provide some or all of the processing resources provided in hardware and/or firmware by an SPU. HPE 655 in one preferred embodiment of the present invention is full-featured and fully compatible with SPE 503—that is, HPE 655 can handle each and every service call SPE 503 can handle such that the SPE and the HPE are "plug compatible" from an outside interface standpoint (with the exception that the HPE may not provide as much security as the SPE).

20

HPEs 655 may be provided in two types: secure and not secure. For example, it may be desirable to provide non-secure versions of HPE 655 to allow electronic appliance 600 to efficiently run non-sensitive VDE tasks using the full resources

of a fast general purpose processor or computer. Such non-secure versions of HPE 655 may run under supervision of an instance of ROS 602 that also includes an SPE 503. In this way, ROS 602 may run all secure processes within SPE 503, and only
5 use HPE 655 for processes that do not require security but that may require (or run more efficiently) under potentially greater resources provided by a general purpose computer or processor supporting HPE 655. Non-secure and secure HPE 655 may operate together with a secure SPE 503.

10

HPEs 655 may (as shown in Figure 10) be provided with a software-based tamper resistant barrier 674 that makes them more secure. Such a software-based tamper resistant barrier 674 may be created by software executing on general-purpose
15 CPU 654. Such a "secure" HPE 655 can be used by ROS 602 to execute processes that, while still needing security, may not require the degree of security provided by SPU 500. This can be especially beneficial in architectures providing both an SPE 503 and an HPE 655. The SPU 502 may be used to perform all truly
20 secure processing, whereas one or more HPEs 655 may be used to provide additional secure (albeit possibly less secure than the SPE) processing using host processor or other general purpose resources that may be available within an electronic appliance 600. Any service may be provided by such a secure HPE 655. In

the preferred embodiment, certain aspects of "channel processing" appears to be a candidate that could be readily exported from SPE 503 to HPE 655.

5 The software-based tamper resistant barrier 674 provided by HPE 655 may be provided, for example, by: introducing time checks and/or code modifications to complicate the process of stepping through code comprising a portion of kernel 688a and/or a portion of component assemblies 690 using a debugger; using a
10 map of defects on a storage device (e.g., a hard disk, memory card, etc.) to form internal test values to impede moving and/or copying HPE 655 to other electronic appliances 600; using kernel code that contains false branches and other complications in flow of control to disguise internal processes to some degree from
15 disassembly or other efforts to discover details of processes; using "self-generating" code (based on the output of a co-sine transform, for example) such that detailed and/or complete instruction sequences are not stored explicitly on storage devices and/or in active memory but rather are generated as needed;
20 using code that "shuffles" memory locations used for data values based on operational parameters to complicate efforts to manipulate such values; using any software and/or hardware memory management resources of electronic appliance 600 to "protect" the operation of HPE 655 from other processes,

functions, etc. Although such a software-based tamper resistant barrier 674 may provide a fair degree of security, it typically will not be as secure as the hardware-based tamper resistant barrier 502 provided (at least in part) by SPU 500. Because security
5 may be better/more effectively enforced with the assistance of hardware security features such as those provided by SPU 500 (and because of other factors such as increased performance provided by special purpose circuitry within SPU 500), at least one SPE 503 is preferred for many or most higher security
10 applications. However, in applications where lesser security can be tolerated and/or the cost of an SPU 500 cannot be tolerated, the SPE 503 may be omitted and all secure processing may instead be performed by one or more secure HPEs 655 executing on general-purpose CPUs 654. Some VDE processes may not be
15 allowed to proceed on reduced-security electronic appliances of this type if insufficient security is provided for the particular process involved.

Only those processes that execute completely within SPEs
20 503 (and in some cases, HPEs 655) may be considered to be truly secure. Memory and other resources external to SPE 503 and HPEs 655 used to store and/or process code and/or data to be used in secure processes should only receive and handle that information in encrypted form unless SPE 503/HPE 655 can

protect secure process code and/or data from non-secure processes.

5 OS "core" 679 in the preferred embodiment includes a kernel 680, an RPC manager 732, and an "object switch" 734. API 682, HPE 655 and SPE 503 may communicate "event" messages with one another via OS "core" 679. They may also communicate messages directly with one another without messages going through OS "core" 679.

10

Kernel 680 may manage the hardware of an electronic appliance 600. For example, it may provide appropriate drivers and hardware managers for interacting with input/output and/or peripheral devices such as keyboard 612, display 614, other
15 devices such as a "mouse" pointing device and speech recognizer 613, modem 618, printer 622, and an adapter for network 672. Kernel 680 may also be responsible for initially loading the remainder of ROS 602, and may manage the various ROS tasks (and associated underlying hardware resources) during
20 execution. OS kernel 680 may also manage and access secure database 610 and file system 687. OS kernel 680 also provides execution services for applications 608a(1), 608a(2), etc. and other applications.

RPC manager 732 performs messaging routing and resource management/integration for ROS 680. It receives and routes "calls" from/to API 682, HPE 655 and SPE 503, for example.

5

Object switch 734 may manage construction, deconstruction and other manipulation of VDE objects 300.

10

User Notification/Exception Interface 686 in the preferred embodiment (which may be considered part of API 682 or another application coupled to the API) provides "pop up" windows/displays on display 614. This allows ROS 602 to communicate directly with a user without having to pass information to be communicated through applications 608. For applications that are not "VDE aware," user notification/exception interface 686 may provide communications between ROS 602 and the user.

15

20

API 682 in the preferred embodiment provides a standardized, documented software interface to applications 608. In part, API 682 may translate operating system "calls" generated by applications 608 into Remote Procedure Calls ("RPCs") specifying "events." RPC manager 732 may route these RPCs to kernel 680 or elsewhere (e.g., to HPE(s) 655 and/or

SPE(s) 503, or to remote electronic appliances 600, processors, or VDE participants) for processing. The API 682 may also service RPC requests by passing them to applications 608 that register to receive and process specific requests.

5

API 682 provides an "Applications Programming Interface" that is preferably standardized and documented. It provides a concise set of function calls an application program can use to access services provided by ROS 602. In at least one preferred example, API 682 will include two parts: an application program interface to VDE functions 604; and an application program interface to other OS functions 606. These parts may be interwoven into the same software, or they may be provided as two or more discrete pieces of software (for example).

10

15

Some applications, such as application 608a(1) shown in Figure 11, may be "VDE aware" and may therefore directly access both of these parts of API 682. Figure 11A shows an example of this. A "VDE aware" application may, for example, include explicit calls to ROS 602 requesting the creation of new VDE objects 300, metering usage of VDE objects, storing information in VDE-protected form, etc. Thus, a "VDE aware" application can initiate (and, in some examples, enhance and/or extend) VDE functionality provided by ROS 602. In addition,

20

"VDE aware" applications may provide a more direct interface between a user and ROS 602 (e.g., by suppressing or otherwise dispensing with "pop up" displays otherwise provided by user notification/exception interface 686 and instead providing a more
5 "seamless" interface that integrates application and ROS messages).

Other applications, such as application 608b shown in Figure 11B, may not be "VDE Aware" and therefore may not
10 "know" how to directly access an interface to VDE functions 604 provided by API 682. To provide for this, ROS 602 may include a "redirector" 684 that allows such "non-VDE aware" applications 608(b) to access VDE objects 300 and functions 604. Redirector 684, in the preferred embodiment, translates OS calls directed to
15 the "other OS functions" 606 into calls to the "VDE functions" 604. As one simple example, redirector 684 may intercept a "file open" call from application 608(b), determine whether the file to be opened is contained within a VDE container 300, and if it is, generate appropriate VDE function call(s) to file system 687 to
20 open the VDE container (and potentially generate events to HPE 655 and/or SPE 503 to determine the name(s) of file(s) that may be stored in a VDE object 300, establish a control structure associated with a VDE object 300, perform a registration for a VDE object 300, etc.). Without redirector 684 in this example, a

non-VDE aware application such as 608b could access only the part of API 682 that provides an interface to other OS functions 606, and therefore could not access any VDE functions.

5 This "translation" feature of redirector 684 provides "transparency." It allows VDE functions to be provided to the application 608(b) in a "transparent" way without requiring the application to become involved in the complexity and details associated with generating the one or more calls to VDE
10 functions 604. This aspect of the "transparency" features of ROS 602 has at least two important advantages:

- (a) it allows applications not written specifically for VDE functions 604 ("non-VDE aware applications") to nevertheless access critical VDE functions; and
- 15 (b) it reduces the complexity of the interface between an application and ROS 602.

Since the second advantage (reducing complexity) makes it easier for an application creator to produce applications, even "VDE aware" applications 608a(2) may be designed so that some
20 calls invoking VDE functions 604 are requested at the level of an "other OS functions" call and then "translated" by redirector 684 into a VDE function call (in this sense, redirector 684 may be considered a part of API 682). Figure 11C shows an example of

this. Other calls invoking VDE functions 604 may be passed directly without translation by redirector 684.

Referring again to Figure 10, ROS 620 may also include an
5 "interceptor" 692 that transmits and/or receives one or more real time data feeds 694 (this may be provided over cable(s) 628 for example), and routes one or more such data feeds appropriately while providing "translation" functions for real time data sent and/or received by electronic appliance 600 to allow
10 "transparency" for this type of information analogous to the transparency provided by redirector 684 (and/or it may generate one or more real time data feeds).

Secure ROS Components and Component Assemblies

15 As discussed above, ROS 602 in the preferred embodiment is a component-based architecture. ROS VDE functions 604 may be based on segmented, independently loadable executable "component assemblies" 690. These component assemblies 690 are independently securely deliverable. The component
20 assemblies 690 provided by the preferred embodiment comprise code and data elements that are themselves independently deliverable. Thus, each component assembly 690 provided by the preferred embodiment is comprised of independently securely deliverable elements which may be communicated using VDE

secure communication techniques, between VDE secure subsystems.

5 These component assemblies 690 are the basic functional unit provided by ROS 602. The component assemblies 690 are executed to perform operating system or application tasks. Thus, some component assemblies 690 may be considered to be part of the ROS operating system 602, while other component assemblies may be considered to be "applications" that run under
10 the support of the operating system. As with any system incorporating "applications" and "operating systems," the boundary between these aspects of an overall system can be ambiguous. For example, commonly used "application" functions (such as determining the structure and/or other attributes of a
15 content container) may be incorporated into an operating system. Furthermore, "operating system" functions (such as task management, or memory allocation) may be modified and/or replaced by an application. A common thread in the preferred embodiment's ROS 602 is that component assemblies 690
20 provide functions needed for a user to fulfill her intended activities, some of which may be "application-like" and some of which may be "operating system-like."

Components 690 are preferably designed to be easily separable and individually loadable. ROS 602 assembles these elements together into an executable component assembly 690 prior to loading and executing the component assembly (e.g., in a secure operating environment such as SPE 503 and/or HPE 655). ROS 602 provides an element identification and referencing mechanism that includes information necessary to automatically assemble elements into a component assembly 690 in a secure manner prior to, and/or during, execution.

ROS 602 application structures and control parameters used to form component assemblies 690 can be provided by different parties. Because the components forming component assemblies 690 are independently securely deliverable, they may be delivered at different times and/or by different parties ("delivery" may take place within a local VDE secure subsystem, that is submission through the use of such a secure subsystem of control information by a chain of content control information handling participant for the preparation of a modified control information set constitutes independent, secure delivery). For example, a content creator can produce a ROS 602 application that defines the circumstances required for licensing content contained within a VDE object 300. This application may reference structures provided by other parties. Such references

might, for example, take the form of a control path that uses content creator structures to meter user activities; and structures created/owned by a financial provider to handle financial parts of a content distribution transaction (e.g.,
5 defining a credit budget that must be present in a control structure to establish creditworthiness, audit processes which must be performed by the licensee, etc.). As another example, a distributor may give one user more favorable pricing than another user by delivering different data elements defining
10 pricing to different users. This attribute of supporting multiple party securely, independently deliverable control information is fundamental to enabling electronic commerce, that is, defining of a content and/or appliance control information set that represents the requirements of a collection of independent
15 parties such as content creators, other content providers, financial service providers, and/or users.

In the preferred embodiment, ROS 602 assembles securely independently deliverable elements into a component assembly
20 690 based in part on context parameters (e.g., object, user). Thus, for example, ROS 602 may securely assemble different elements together to form different component assemblies 690 for different users performing the same task on the same VDE object 300. Similarly, ROS 602 may assemble differing element

sets which may include, that is reuse, one or more of the same components to form different component assemblies 690 for the same user performing the same task on different VDE objects 300.

5

The component assembly organization provided by ROS 602 is "recursive" in that a component assembly 690 may comprise one or more component "subassemblies" that are themselves independently loadable and executable component assemblies 690. These component "subassemblies" may, in turn, be made of one or more component "sub-sub-assemblies." In the general case, a component assembly 690 may include N levels of component subassemblies.

10

Thus, for example, a component assembly 690(k) that may includes a component subassembly 690(k + 1). Component subassembly 690(k + 1), in turn, may include a component sub-sub-assembly 690(3), ... and so on to N-level subassembly 690(k + N). The ability of ROS 602 to build component assemblies 690 out of other component assemblies provides great advantages in terms of, for example, code/data reusability, and the ability to allow different parties to manage different parts of an overall component.

20

Each component assembly 690 in the preferred embodiment is made of distinct components. Figures 11D-11H are abstract depictions of various distinct components that may be assembled to form a component assembly 690(k) showing
5 Figure 11I. These same components can be combined in different ways (e.g., with more or less components) to form different component assemblies 690 providing completely different functional behavior. Figure 11J is an abstract depiction of the same components being put together in a different way
10 (e.g., with additional components) to form a different component assembly 690(j). The component assemblies 690(k) and 690(j) each include a common feature 691 that interlocks with a "channel" 594 defined by ROS 602. This "channel" 594 assembles component assemblies 690 and interfaces them with
15 the (rest of) ROS 602.

ROS 602 generates component assemblies 690 in a secure manner. As shown graphically in Figures 11I and 11J, the different elements comprising a component assembly 690 may be
20 "interlocking" in the sense that they can only go together in ways that are intended by the VDE participants who created the elements and/or specified the component assemblies. ROS 602 includes security protections that can prevent an unauthorized person from modifying elements, and also prevent an

5 unauthorized person from substituting elements. One can
picture an unauthorized person making a new element having
the same "shape" as the one of the elements shown in Figures
11D-11H, and then attempting to substitute the new element in
place of the original element. Suppose one of the elements
shown in Figure 11H establishes the price for using content
within a VDE object 300. If an unauthorized person could
substitute her own "price" element for the price element intended
by the VDE content distributor, then the person could establish a
10 price of zero instead of the price the content distributor intended
to charge. Similarly, if the element establishes an electronic
credit card, then an ability to substitute a different element
could have disastrous consequences in terms of allowing a person
to charge her usage to someone else's (or a non-existent) credit
15 card. These are merely a few simple examples demonstrating
the importance of ROS 602 ensuring that certain component
assemblies 690 are formed in a secure manner. ROS 602
provides a wide range of protections against a wide range of
"threats" to the secure handling and execution of component
20 assemblies 690.

In the preferred embodiment, ROS 602 assembles
component assemblies 690 based on the following types of
elements:

Permissions Records ("PERC"s) 808;
Method "Cores" 1000;
Load Modules 1100;
Data Elements (e.g., User Data Elements ("UDEs") 1200
5 and Method Data Elements ("MDEs") 1202); and
Other component assemblies 690.

Briefly, a PERC 808 provided by the preferred
embodiment is a record corresponding to a VDE object 300 that
10 identifies to ROS 602. among other things, the elements ROS is
to assemble together to form a component assembly 690. Thus
PERC 808 in effect contains a "list of assembly instructions" or a
"plan" specifying what elements ROS 602 is to assemble together
into a component assembly and how the elements are to be
15 connected together. PERC 808 may itself contain data or other
elements that are to become part of the component assembly 690.

The PERC 808 may reference one or more method "cores"
1000'. A method core 1000' may define a basic "method" 1000
20 (e.g., "control," "billing," "metering," etc.)

In the preferred embodiment, a "method" 1000 is a
collection of basic instructions, and information related to basic
instructions, that provides context, data, requirements, and/or

relationships for use in performing, and/or preparing to perform, basic instructions in relation to the operation of one or more electronic appliances 600. Basic instructions may be comprised of, for example:

5

- machine code of the type commonly used in the programming of computers; pseudo-code for use by an interpreter or other instruction processing program operating on a computer;
- 10 • a sequence of electronically represented logical operations for use with an electronic appliance 600;
- or other electronic representations of instructions, source code, object code, and/or pseudo code as those terms are commonly understood in the arts.

15

Information relating to said basic instructions may comprise, for example, data associated intrinsically with basic instructions such as for example, an identifier for the combined basic instructions and intrinsic data, addresses, constants, and/or the like. The information may also, for example, include

20 one or more of the following:

- information that identifies associated basic instructions and said intrinsic data for access, correlation and/or validation purposes;
- required and/or optional parameters for use with
5 basic instructions and said intrinsic data;
- information defining relationships to other methods;
- data elements that may comprise data values, fields of information, and/or the like;
- information specifying and/or defining relationships
10 among data elements, basic instructions and/or intrinsic data;
- information specifying relationships to external data elements;
- information specifying relationships between and
15 among internal and external data elements, methods, and/or the like, if any exist; and
- additional information required in the operation of basic instructions and intrinsic data to complete, or
20 attempt to complete, a purpose intended by a user of a method, where required, including additional instructions and/or intrinsic data.

Such information associated with a method may be stored, in part or whole, separately from basic instructions and intrinsic data. When these components are stored separately, a method may nevertheless include and encompass the other information and one or more sets of basic instructions and intrinsic data (the latter being included because of said other information's reference to one or more sets of basic instructions and intrinsic data), whether or not said one or more sets of basic instructions and intrinsic data are accessible at any given point in time.

10

Method core 1000' may be parameterized by an "event code" to permit it to respond to different events in different ways. For example, a METER method may respond to a "use" event by storing usage information in a meter data structure. The same METER method may respond to an "administrative" event by reporting the meter data structure to a VDE clearinghouse or other VDE participant.

15

In the preferred embodiment, method core 1000' may "contain," either explicitly or by reference, one or more "load modules" 1100 and one or more data elements (UDEs 1200, MDEs 1202). In the preferred embodiment, a "load module" 1100 is a portion of a method that reflects basic instructions and intrinsic data. Load modules 1100 in the preferred embodiment

20

contain executable code, and may also contain data elements ("DTDs" 1108) associated with the executable code. In the preferred embodiment, load modules 1100 supply the program instructions that are actually "executed" by hardware to perform the process defined by the method. Load modules 1100 may contain or reference other load modules.

Load modules 1100 in the preferred embodiment are modular and "code pure" so that individual load modules may be reenterable and reusable. In order for components 690 to be dynamically updatable, they may be individually addressable within a global public name space. In view of these design goals, load modules 1100 are preferably small, code (and code-like) pure modules that are individually named and addressable. A single method may provide different load modules 1100 that perform the same or similar functions on different platforms, thereby making the method scalable and/or portable across a wide range of different electronic appliances.

UDEs 1200 and MDEs 1202 may store data for input to or output from executable component assembly 690 (or data describing such inputs and/or outputs). In the preferred embodiment, UDEs 1200 may be user dependent, whereas MDEs 1202 may be user independent.

5 The component assembly example 690(k) shown in Figure 11E comprises a method core 1000', UDEs 1200a & 1200b, an MDE 1202, load modules 1100a-1100d, and a further component assembly 690(k+1). As mentioned above, a PERC 808(k) defines, among other things, the "assembly instructions" for component assembly 690(k), and may contain or reference parts of some or all of the components that are to be assembled to create a component assembly.

10 One of the load modules 1100b shown in this example is itself comprised of plural load modules 1100c, 1100d. Some of the load modules (e.g., 1100a, 1100d) in this example include one or more "DTD" data elements 1108 (e.g., 1108a, 1108b). "DTD" data elements 1108 may be used, for example, to inform load
15 module 1100a of the data elements included in MDE 1202 and/or UDEs 1200a, 1200b. Furthermore, DTDs 1108 may be used as an aspect of forming a portion of an application used to inform a user as to the information required and/or manipulated by one or more load modules 1100, or other component elements. Such an
20 application program may also include functions for creating and/or manipulating UDE(s) 1200, MDE(s) 1202, or other component elements, subassemblies, etc.

Components within component assemblies 690 may be "reused" to form different component assemblies. As mentioned above, figure 11F is an abstract depiction of one example of the same components used for assembling component assembly 690(k) to be reused (e.g., with some additional components specified by a different set of "assembly instructions" provided in a different PERC 808(1)) to form a different component assembly 690(l). Even though component assembly 690(l) is formed from some of the same components used to form component assembly 690(k), these two component assemblies may perform completely different processes in complete different ways.

As mentioned above, ROS 602 provides several layers of security to ensure the security of component assemblies 690. One important security layer involves ensuring that certain component assemblies 690 are formed, loaded and executed only in secure execution space such as provided within an SPU 500. Components 690 and/or elements comprising them may be stored on external media encrypted using local SPU 500 generated and/or distributor provided keys.

ROS 602 also provides a tagging and sequencing scheme that may be used within the loadable component assemblies 690 to detect tampering by substitution. Each element comprising a

component assembly 690 may be loaded into an SPU 500,
decrypted using encrypt/decrypt engine 522, and then
tested/compared to ensure that the proper element has been
loaded. Several independent comparisons may be used to ensure
5 there has been no unauthorized substitution. For example, the
public and private copies of the element ID may be compared to
ensure that they are the same, thereby preventing gross
substitution of elements. In addition, a validation/correlation
tag stored under the encrypted layer of the loadable element may
10 be compared to make sure it matches one or more tags provided
by a requesting process. This prevents unauthorized use of
information. As a third protection, a device assigned tag (e.g., a
sequence number stored under an encryption layer of a loadable
element may be checked to make sure it matches a corresponding
15 tag value expected by SPU 500. This prevents substitution of
older elements. Validation/correlation tags are typically passed
only in secure wrappers to prevent plaintext exposure of this
information outside of SPU 500.

20 The secure component based architecture of ROS 602 has
important advantages. For example, it accommodates limited
resource execution environments such as provided by a lower
cost SPU 500. It also provides an extremely high level of
configurability. In fact, ROS 602 will accommodate an almost

unlimited diversity of content types, content provider objectives,
transaction types and client requirements. In addition, the
ability to dynamically assemble independently deliverable
components at execution time based on particular objects and
5 users provides a high degree of flexibility, and facilitates or
enables a distributed database, processing, and execution
environment.

One aspect of an advantage of the component-based
10 architecture provided by ROS 602 relates to the ability to "stage"
functionality and capabilities over time. As designed,
implementation of ROS 602 is a finite task. Aspects of its wealth
of functionality can remain unexploited until market realities
dictate the implementation of corresponding VDE application
15 functionality. As a result, initial product implementation
investment and complexity may be limited. The process of
"surfacing" the full range of capabilities provided by ROS 602 in
terms of authoring, administrative, and artificial intelligence
applications may take place over time. Moreover, already-
20 designed functionality of ROS 602 may be changed or enhanced
at any time to adapt to changing needs or requirements.

More Detailed Discussion of Rights Operating System 602 Architecture

5 Figure 12 shows an example of a detailed architecture of
ROS 602 shown in Figure 10. ROS 602 may include a file system
687 that includes a commercial database manager 730 and
external object repositories 728. Commercial database manager
730 may maintain secure database 610. Object repository 728
may store, provide access to, and/or maintain VDE objects 300.

10

 Figure 12 also shows that ROS 602 may provide one or
more SPEs 503 and/or one or more HPEs 655. As discussed
above, HPE 655 may "emulate" an SPU 500 device, and such
HPEs 655 may be integrated in lieu of (or in addition to) physical
15 SPUs 500 for systems that need higher throughput. Some
security may be lost since HPEs 655 are typically protected by
operating system security and may not provide truly secure
processing. Thus, in the preferred embodiment, for high security
applications at least, all secure processing should take place
20 within an SPE 503 having an execution space within a physical
SPU 500 rather than a HPE 655 using software operating
elsewhere in electronic appliance 600.

 As mentioned above, three basic components of ROS 602
25 are a kernel 680, a Remote Procedure Call (RPC) manager 732

and an object switch 734. These components, and the way they interact with other portions of ROS 602, will be discussed below.

Kernel 680

5 Kernel 680 manages the basic hardware resources of electronic appliance 600, and controls the basic tasking provided by ROS 602. Kernel 680 in the preferred embodiment may include a memory manager 680a, a task manager 680b, and an I/O manager 680c. Task manager 680b may initiate and/or
10 manage initiation of executable tasks and schedule them to be executed by a processor on which ROS 602 runs (e.g., CPU 654 shown in Figure 8). For example, Task manager 680b may include or be associated with a "bootstrap loader" that loads other parts of ROS 602. Task manager 680b may manage all
15 tasking related to ROS 602, including tasks associated with application program(s) 608. Memory manager 680a may manage allocation, deallocation, sharing and/or use of memory (e.g., RAM 656 shown in Figure 8) of electronic appliance 600, and may for example provide virtual memory capabilities as required by an
20 electronic appliance and/or associated application(s). I/O manager 680c may manage all input to and output from ROS 602, and may interact with drivers and other hardware managers that provide communications and interactivity with physical devices.

RPC Manager 732

ROS 602 in a preferred embodiment is designed around a "services based" Remote Procedure Call architecture/interface. All functions performed by ROS 602 may use this common interface to request services and share information. For example, SPE(s) 503 provide processing for one or more RPC based services. In addition to supporting SPU's 500, the RPC interface permits the dynamic integration of external services and provides an array of configuration options using existing operating system components. ROS 602 also communicates with external services through the RPC interface to seamlessly provide distributed and/or remote processing. In smaller scale instances of ROS 602, a simpler message passing IPC protocol may be used to conserve resources. This may limit the configurability of ROS 602 services, but this possible limitation may be acceptable in some electronic appliances.

The RPC structure allows services to be called/requested without the calling process having to know or specify where the service is physically provided, what system or device will service the request, or how the service request will be fulfilled. This feature supports families of services that may be scaled and/or customized for specific applications. Service requests can be forwarded and serviced by different processors and/or different

sites as easily as they can be forwarded and serviced by a local service system. Since the same RPC interface is used by ROS 602 in the preferred embodiment to request services within and outside of the operating system, a request for distributed and/or remote processing incurs substantially no additional operating system overhead. Remote processing is easily and simply integrated as part of the same service calls used by ROS 602 for requesting local-based services. In addition, the use of a standard RPC interface ("RSI") allows ROS 602 to be modularized, with the different modules presenting a standardized interface to the remainder of the operating system. Such modularization and standardized interfacing permits different vendors operating system programmers to create different portions of the operating system independently, and also allows the functionality of ROS 602 to be flexibly updated and/or changed based on different requirements and/or platforms.

RPC manager 732 manages the RPC interface. It receives service requests in the form of one or more "Remote Procedure Calls" (RPCs) from a service requestor, and routes the service requests to a service provider(s) that can service the request. For example, when rights operating system 602 receives a request from a user application via user API 682, RPC manager 732 may

route the service request to an appropriate service through the "RPC service interface" ("RSI"). The RSI is an interface between RPC manager 732, service requestors, and a resource that will accept and service requests.

5

The RPC interface (RSI) is used for several major ROS 602 subsystems in the preferred embodiment.

RPC services provided by ROS 602 in the preferred
10 embodiment are divided into subservices, i.e., individual instances of a specific service each of which may be tracked individually by the RPC manager 732. This mechanism permits multiple instances of a specific service on higher throughput systems while maintaining a common interface across a
15 spectrum of implementations. The subservice concept extends to supporting multiple processors, multiple SPEs 503, multiple HPEs 655, and multiple communications services.

The preferred embodiment ROS 602 provides the following
20 RPC based service providers/requestors (each of which have an RPC interface or "RSI" that communicates with RPC manager 732):

SPE device driver 736 (this SPE device driver is connected to an SPE 503 in the preferred embodiment);

HPE Device Driver 738 (this HPE device driver is connected to an HPE 738 in the preferred embodiment);

Notification Service 740 (this notification service is connected to user notification interface 686 in the preferred embodiment);

API Service 742 (this API service is connected to user API 682 in the preferred embodiment);

Redirector 684;

Secure Database (File) Manager 744 (this secure database or file manager 744 may connect to and interact with commercial database manager 730 and secure files 610 through a cache manager 746, a database interface 748, and a database driver 750);

Name Services Manager 752;

Outgoing Administrative Objects Manager 754;

Incoming Administrative Objects Manager 756;

a Gateway 734 to object switch 734 (this is a path used to allow direct communication between RPC manager

732 and Object Switch 734); and

Communications Manager 776.

The types of services provided by HPE 655, SPE 503, User Notification 686, API 742 and Redirector 684 have already been

described above. Here is a brief description of the type(s) of services provided by OS resources 744, 752, 754, 756 and 776:

Secure Database Manager 744 services requests for access to secure database 610;

5 Name Services Manager 752 services requests relating to user, host, or service identification;

Outgoing Administrative Objects Manager 754 services requests relating to outgoing administrative objects;

10 Incoming Administrative Objects Manager 756 services requests relating to incoming administrative objects; and

Communications Manager 776 services requests relating to communications between electronic appliance 600 and the outside world.

15

Object Switch 734

Object switch 734 handles, controls and communicates (both locally and remotely) VDE objects 300. In the preferred embodiment, the object switch may include the following elements:

20

- a stream router 758;
- a real time stream interface(s) 760 (which may be connected to real time data feed(s) 694);
- a time dependent stream interface(s) 762;

a intercept 692;
a container manager 764;
one or more routing tables 766; and
buffering/storage 768.

- 5 Stream router 758 routes to/from "real time" and "time independent" data streams handled respectively by real time stream interface(s) 760 and time dependent stream interface(s) 762. Intercept 692 intercepts I/O requests that involve real-time information streams such as, for example, real time feed 694.
- 10 The routing performed by stream router 758 may be determined by routing tables 766. Buffering/storage 768 provides temporary store-and-forward, buffering and related services. Container manager 764 may (typically in conjunction with SPE 503) perform processes on VDE objects 300 such as constructing,
- 15 deconstructing, and locating portions of objects.

Object switch 734 communicates through an Object Switch Interface ("OSI") with other parts of ROS 602. The Object Switch Interface may resemble, for example, the interface for a

20 Unix socket in the preferred embodiment. Each of the "OSI" interfaces shown in Figure 12 have the ability to communicate with object switch 734.

ROS 602 includes the following object switch service providers/resources (each of which can communicate with the object switch 734 through an "OSI"):

Outgoing Administrative Objects Manager 754;

5 Incoming Administrative Objects Manager 756;

Gateway 734 (which may translate RPC calls into object switch calls and vice versa so RPC manager 732 may communicate with object switch 734 or any other element having an OSI to, for example, provide and/or request services);

10

External Services Manager 772;
Object Submittal Manager 774; and
Communications Manager 776.

15

Briefly,

Object Repository Manager 770 provides services relating to access to object repository 728;

External Services Manager 772 provides services relating to requesting and receiving services externally, such as from a network resource or another site;

20

Object Submittal Manager 774 provides services relating to how a user application may interact with object switch 734 (since the object submittal manager

provides an interface to an application program 608,
it could be considered part of user API 682); and

Communications Manager 776 provides services relating
to communicating with the outside world.

5

In the preferred embodiment, communications manager
776 may include a network manager 780 and a mail gateway
(manager) 782. Mail gateway 782 may include one or more mail
filters 784 to, for example, automatically route VDE related
electronic mail between object switch 734 and the outside world
electronic mail services. External Services Manager 772 may
interface to communications manager 776 through a Service
Transport Layer 786. Service Transport Layer 786a may enable
External Services Manager 772 to communicate with external
computers and systems using various protocols managed using
the service transport layer 786.

10

15

The characteristics of and interfaces to the various
subsystems of ROS 680 shown in Figure 12 are described in more
detail below.

20

RPC Manager 732 and Its RPC Services Interface

As discussed above, the basic system services provided by
ROS 602 are invoked by using an RPC service interface (RSI).

This RPC service interface provides a generic, standardized interface for different services systems and subsystems provided by ROS 602.

5 RPC Manager 732 routes RPCs requesting services to an appropriate RPC service interface. In the preferred embodiment, upon receiving an RPC call, RPC manager 732 determines one or more service managers that are to service the request. RPC manager 732 then routes a service request to the appropriate
10 service(s) (via a RSI associated with a service) for action by the appropriate service manager(s).

For example, if a SPE 503 is to service a request, the RPC Manager 732 routes the request to RSI 736a, which passes the
15 request on to SPE device driver 736 for forwarding to the SPE. Similarly, if HPE 655 is to service the request, RPC Manager 732 routes the request to RSI 738a for forwarding to a HPE. In one preferred embodiment, SPE 503 and HPE 655 may perform essentially the same services so that RSIs 736a, 738a are
20 different instances of the same RSI. Once a service request has been received by SPE 503 (or HPE 655), the SPE (or HPE) typically dispatches the request internally using its own internal RPC manager (as will be discussed shortly). Processes within SPEs 503 and HPEs 655 can also generate RPC requests. These

requests may be processed internally by a SPE/HPE, or if not internally serviceable, passed out of the SPE/HPE for dispatch by RPC Manager 732.

5 Remote (and local) procedure calls may be dispatched by a
RPC Manager 732 using an "RPC Services Table." An RPC
Services Table describes where requests for specific services are
to be routed for processing. Each row of an RPC Services Table
in the preferred embodiment contains a services ID, the location
10 of the service, and an address to which control will be passed to
service a request. An RPC Services Table may also include
control information that indicates which instance of the RPC
dispatcher controls the service. Both RPC Manager 732 and any
attached SPEs 503 and HPEs 655 may have symmetric copies of
15 the RPC Services Table. If an RPC service is not found in the
RPC services tables, it is either rejected or passed to external
services manager 772 for remote servicing.

20 Assuming RPC manager 732 finds a row corresponding to
the request in an RPC Services Table, it may dispatch the
request to an appropriate RSI. The receiving RSI accepts a
request from the RPC manager 732 (which may have looked up
the request in an RPC service table), and processes that request

in accordance with internal priorities associated with the specific service.

5 In the preferred embodiment, RPC Service Interface(s) supported by RPC Manager 732 may be standardized and published to support add-on service modules developed by third party vendors, and to facilitate scalability by making it easier to program ROS 602. The preferred embodiment RSI closely follows the DOS and Unix device driver models for block devices
10 so that common code may be developed for many platforms with minimum effort. An example of one possible set of common entry points are listed below in the table.

15

20

Interface call	Description
SVC_LOAD	Load a service manager and return its status.
SVC_UNLOAD	Unload a service manager.
SVC_MOUNT	Mount (load) a dynamically loaded subservice and return its status.
SVC_UNMOUNT	Unmount (unload) a dynamically loaded subservice.
SVC_OPEN	Open a mounted subservice.
SVC_CLOSE	Close a mounted subservice.
SVC_READ	Read a block from an opened subservice.

SVC_WRITE	Write a block to an opened subservice.
SVC_IOCTL	Control a subservice or a service manager.

Load

5 In the preferred embodiment, services (and the associated RSIs they present to RPC manager 732) may be activated during boot by an installation boot process that issues an RPC LOAD. This process reads an RPC Services Table from a configuration file, loads the service module if it is run time loadable (as
10 opposed to being a kernel linked device driver), and then calls the LOAD entry point for the service. A successful return from the LOAD entry point will indicate that the service has properly loaded and is ready to accept requests.

15 RPC LOAD Call Example: SVC_LOAD (long service_id)

 This LOAD interface call is called by the RPC manager 732 during rights operating system 602 initialization. It permits a service manager to load any dynamically loadable components and to initialize any device and memory required by the service.
20 The service number that the service is loaded as is passed in as *service_id* parameter. In the preferred embodiment, the service

returns 0 is the initialization process was completed successfully or an error number if some error occurred.

Mount

5 Once a service has been loaded, it may not be fully functional for all subservices. Some subservices (e.g., communications based services) may require the establishment of additional connections, or they may require additional modules to be loaded. If the service is defined as "mountable," a
10 RPC manager 732 will call the MOUNT subservice entry point with the requested subservice ID prior to opening an instance of a subservice.

RPC MOUNT Call Example:

15 SVC_MOUNT (long service_id, long subservice_id, BYTE *buffer)

20 This MOUNT interface call instructs a service to make a specific subservice ready. This may include services related to networking, communications, other system services, or external resources. The *service_id* and *subservice_id* parameters may be specific to the specific service being requested. The *buffer* parameter is a memory address that references a control structure appropriate to a specific service.

Open

Once a service is loaded and "mounted," specific instances of a service may be "opened" for use. "Opening" an instance of a service may allocate memory to store control and status information. For example, in a BSD socket based network connection, a LOAD call will initialize the software and protocol control tables, a MOUNT call will specify networks and hardware resources, and an OPEN will actually open a socket to a remote installation.

10

Some services, such as commercial database manager 730 that underlies the secure database service, may not be "mountable." In this case, a LOAD call will make a connection to a database manager 730 and ensure that records are readable. An OPEN call may create instances of internal cache manager 746 for various classes of records.

15

RPC OPEN Call Example:

SVC_OPEN (long service_id, long subservice_id, BYTE
*buffer, int (*receive) (long request_id))

20

This OPEN interface call instructs a service to open a specific subservice. The *service_id* and *subservice_id* parameters are specific to the specific service being requested,

and the *buffer* parameter is a memory address that references a control structure appropriate to a specific service.

5 The optional *receive* parameter is the address of a notification callback function that is called by a service whenever a message is ready for the service to retrieve it. One call to this address is made for each incoming message received. If the caller passes a NULL to the interface, the software will not generate a callback for each message.

10

Close, Unmount and Unload

15 The converse of the OPEN, MOUNT, and LOAD calls are CLOSE, UNMOUNT, and UNLOAD. These interface calls release any allocated resources back to ROS 602 (e.g., memory manager 680a).

RPC CLOSE Call Example: SVC_CLOSE (long svc_handle)

20 This LOAD interface call closes an open service "handle." A service "handle" describes a service and subservice that a user wants to close. The call returns 0 if the CLOSE request succeeds (and the handle is no longer valid) or an error number.

RPC UNLOAD Call Example: SVC_UNLOAD (void)

This UNLOAD interface call is called by a RPC manager 732 during shutdown or resource reallocation of rights operating system 602. It permits a service to close any open connections, flush buffers, and to release any operating system resources that it may have allocated. The service returns 0.

RPC UNMOUNT Call Example: SVC_UNMOUNT (long service_id, long subservice_id)

This UNMOUNT interface call instructs a service to deactivate a specific subservice. The *service_id* and *subservice_id* parameters are specific to the specific service being requested, and must have been previously mounted using the *SVC_MOUNT()* request. The call releases all system resources associated with the subservice before it returns.

Read and Write

The READ and WRITE calls provide a basic mechanism for sending information to and receiving responses from a mounted and opened service. For example, a service has requests written to it in the form of an RPC request, and makes its response available to be read by RPC Manager 732 as they become available.

RPC READ Call Example:

SVC_READ (long svc_handle, long request_id, BYTE
*buffer, long size)

This READ call reads a message response from a service.

5 The *svc_handle* and *request_id* parameters uniquely identify a request. The results of a request will be stored in the user specified *buffer* up to *size* bytes. If the buffer is too small, the first size bytes of the message will be stored in the buffer and an error will be returned.

10

If a message response was returned to the caller's buffer correctly, the function will return 0. Otherwise, an error message will be returned.

15 **RPC WRITE Call Example:**

SVC_write (long service_id, long subservice_id, BYTE
*buffer, long size, int (*receive) (long request_id))

20 This WRITE call writes a message to a service and subservice specified by the *service_id/subservice_id* parameter pair. The message is stored in buffer (and usually conforms to the VDE RPC message format) and is *size* bytes long. The function returns the *request id* for the message (if it was accepted for sending) or an error number. If a user specifies the *receive* callback functions, all messages regarding a request will

be sent to the request specific callback routine instead of the generalized message callback.

Input/Output Control

5 The IOCTL ("Input/Output ConTroL") call provides a mechanism for querying the status of and controlling a loaded service. Each service type will respond to specific general IOCTL requests, all required class IOCTL requests, and service specific IOCTL requests.

10

RPC IOCTL Call Example: ROI_SVC_IOCTL (long service_id,
long subservice_id,

int command, BYTE *buffer)

15

This IOCTL function provides a generalized control interface for a RSI. A user specifies the *service_id* parameter and an optional *subservice_id* parameter that they wish to control. They specify the control *command* parameter(s), and a *buffer* into/from which the *command* parameters may be
20 written/read. An example of a list of commands and the appropriate buffer structures are given below.

Command	Structure	Description
GET_INFO	SVC_INFO	Returns information about a service/subservice.
GET_STATS	SVC_STATS	Returns current statistics about a service/subservice.
CLR_STATS	None	Clears the statistics about a service/subservice.

* * * * *

Now that a generic RPC Service Interface provided by the preferred embodiment has been described, the following description relates to particular examples of services provided by ROS 602.

SPE Device Driver 736

SPE device driver 736 provides an interface between ROS 602 and SPE 503. Since SPE 503 in the preferred embodiment runs within the confines of an SPU 500, one aspect of this device driver 736 is to provide low level communications services with the SPU 500 hardware. Another aspect of SPE device driver 736 is to provide an RPC service interface (RSI) 736a particular to

SPE 503 (this same RSI may be used to communicate with HPE 655 through HPE device driver 738).

5 SPE RSI 736a and driver 736 isolates calling processes within ROS 602 (or external to the ROS) from the detailed service provided by the SPE 503 by providing a set of basic interface points providing a concise function set. This has several advantages. For example, it permits a full line of scaled SPU 500 that all provide common functionality to the outside world but which may differ in detailed internal structure and architecture. SPU 500 characteristics such as the amount of memory resident in the device, processor speed, and the number of services supported within SPU 500 may be the decision of the specific SPU manufacturer, and in any event may differ from one 10 SPU configuration to another. To maintain compatibility, SPE device driver 736 and the RSI 736a it provides conform to a basic common RPC interface standard that "hides" differences between detailed configurations of SPU 500 and/or the SPEs-503 they may support.

20

To provide for such compatibility, SPE RSI 736a in the preferred embodiment follows a simple block based standard. In the preferred embodiment, an SPE RSI 736a may be modeled after the packet interfaces for network Ethernet cards. This

standard closely models the block mode interface characteristics of SPU's 500 in the preferred embodiment.

5 An SPE RSI 736a allows RPC calls from RPC manager 732 to access specific services provided by an SPE 736. To do this, SPE RSI 736a provides a set of "service notification address interfaces." These provide interfaces to individual services provided by SPE 503 to the outside world. Any calling process within ROS 602 may access these SPE-provided services by
10 directing an RPC call to SPE RSI 736a and specifying a corresponding "service notification address" in an RPC call. The specified "service notification address" causes SPE 503 to internally route an RPC call to a particular service within an SPE. The following is a listing of one example of a SPE service
15 breakdown for which individual service notification addresses may be provided:

Channel Services Manager

Authentication Manager/Secure Communications Manager

Secure Database Manager

20

The Channel Services Manager is the principal service provider and access point to SPE 503 for the rest of ROS 602. Event processing, as will be discussed later, is primarily managed (from the point of view of processes outside SPE 503)

by this service. The Authentication Manager/Secure Communications Manager may provide login/logout services for users of ROS 602, and provide a direct service for managing communications (typically encrypted or otherwise protected) related to component assemblies 690, VDE objects 300, etc. Requests for display of information (e.g., value remaining in a financial budget) may be provided by a direct service request to a Secure Database Manager inside SPE 503. The instances of Authentication Manager/Secure Communications Manager and Secure Database Manager, if available at all, may provide only a subset of the information and/or capabilities available to processes operating inside SPE 503. As stated above, most (potentially all) service requests entering SPE are routed to a Channel Services Manager for processing. As will be discussed in more detail later on, most control structures and event processing logic is associated with component assemblies 690 under the management of a Channel Services Manager.

The SPE 503 must be accessed through its associated SPE driver 736 in this example. Generally, calls to SPE driver 736 are made in response to RPC calls. In this example, SPE driver RSI 736a may translate RPC calls directed to control or ascertain information about SPE driver 736 into driver calls. SPE driver

RSI 736a in conjunction with driver 736 may pass RPC calls directed to SPE 503 through to the SPE.

The following table shows one example of SPE device

5 driver 736 calls:

Entry Point	Description
SPE_info()	Returns summary information about the SPE driver 736 (and SPE 503)
SPE_initialize_interface()	Initializes SPE driver 736, and sets the default notification address for received packets.
SPE_terminate_interface()	Terminates SPE driver 736 and resets SPU 500 and the driver 736.
10 SPE_reset_interface()	Resets driver 736 without resetting SPU 500.
SPE_get_stats()	Return statistics for notification addresses and/or an entire driver 736.
SPE_clear_stats()	Clears statistics for a specific notification address and/or an entire driver 736.
SPE_set_notify()	Sets a notification address for a specific service ID.
SPE_get_notify()	Returns a notification address for a specific service ID.
15 SPE_tx_pkt()	Sends a packet (e.g., containing an RPC call) to SPE 503 for processing.

The following are more detailed examples of each of the SPE driver calls set forth in the table above.

Example of an "SPE Information" Driver Call: SPE_info (void)

5 This function returns a pointer to an SPE_INFO data structure that defines the SPE device driver 736a. This data structure may provide certain information about SPE device driver 736, RSI 736a and/or SPU 500. An example of a SPE_INFO structure is described below:

10

15

20

Version Number/ID for SPE Device Driver 736
Version Number/ID for SPE Device Driver RSI 736
Pointer to name of SPE Device Driver 736
Pointer to ID name of SPU 500
Functionality Code Describing SPE Capabilities/functionality

Example of an SPE "Initialize Interface" Driver Call:

SPE_initialize_interface (int (fcn *receiver)(void))

25

5 A receiver function passed in by way of a parameter will be called for all packets received from SPE 503 unless their destination service is over-ridden using the *set_notify()* call. A receiver function allows ROS 602 to specify a format for packet communication between RPC manager 732 and SPE 503.

10 This function returns "0" in the preferred embodiment if the initialization of the interface succeeds and non-zero if it fails. If the function fails, it will return a code that describes the reason for the failure as the value of the function.

Example of an SPE "Terminate Interface" Driver Call:

SPE_terminate_interface (void)

15 In the preferred embodiment, this function shuts down SPE Driver 736, clears all notification addresses, and terminates all outstanding requests between an SPE and an ROS RPC manager 732. It also resets an SPE 503 (e.g., by a warm reboot of SPU 500) after all requests are resolved.

20 Termination of driver 736 should be performed by ROS 602 when the operating system is starting to shut down. It may also be necessary to issue this call if an SPE 503 and ROS 602 get so far out of synchronization that all processing in an SPE must be reset to a known state.

Example of an SPE "Reset Interface" Driver Call:**SPE_reset_interface (void)**

5 This function resets driver 736, terminates all outstanding
requests between SPE 503 and an ROS RPC manager 732, and
clears all statistics counts. It does not reset the SPU 500, but
simply restores driver 736 to a known stable state.

Example of an SPE "Get Statistics" Driver Call: SPE_get_stats

10 (long service_id)

 This function returns statistics for a specific service
notification interface or for the SPE driver 736 in general. It
returns a pointer to a static buffer that contains these statistics
or NULL if statistics are unavailable (either because an interface
15 is not initialized or because a receiver address was not specified).
An example of the SPE_STATS structure may have the following
definition:

20

Service id
packets rx
packets tx
bytes rx
bytes tx
errors rx

25

5

errors tx
requests tx
req tx completed
req tx cancelled
req rx
req rx completed
req rx cancelled

10

If a user specifies a service ID, statistics associated with packets sent by that service are returned. If a user specified 0 as the parameter, the total packet statistics for the interface are returned.

15

Example of an SPE "Clear Statistics" Driver Call:

SPE_clear_stats (long service_id)

20

This function clears statistics associated with the SPE service_id specified. If no service_id is specified (i.e., the caller passes in 0), global statistics will be cleared. The function returns 0 if statistics are successfully cleared or an error number if an error occurs.

Example of an SPE "Set Notification Address" Driver Call:

SPE_set_notify (long service_id, int (fcn*receiver) (void))

5 This function sets a notification address (receiver) for a specified service. If the notification address is set to NULL, SPE device driver 736 will send notifications for packets to the specified service to the default notification address.

Example of a SPE "Get Notification Address" Driver Call:

SPE_get_notify (long service_id)

10 This function returns a notification address associated with the named service or NULL if no specific notification address has been specified.

Example of an SPE "Send Packet" Driver Call:

15 send_pkt (BYTE *buffer, long size, int (far *receive) (void))

 This function sends a packet stored in buffer of "length" size. It returns 0 if the packet is sent successfully, or returns an error code associated with the failure.

20 **Redirector Service Manager 684**

 The redirector 684 is a piece of systems integration software used principally when ROS 602 is provided by "adding on" to a pre-existing operating system or when "transparent" operation is desired for some VDE functions, as described earlier.

In one embodiment the kernel 680, part of communications manager 776, file system 687, and part of API service 742 may be part of a pre-existing operating system such as DOS, Windows, UNIX, Macintosh System, OS9, PSOS, OS/2, or other
5 operating system platform. The remainder of ROS 602 subsystems shown in Figure 12 may be provided as an "add on" to a preexisting operating system. Once these ROS subsystems have been supplied and "added on," the integrated whole comprises the ROS 602 shown in Figure 12.

10

In a scenario of this type of integration, ROS 602 will continue to be supported by a preexisting OS kernel 680, but may supplement (or even substitute) many of its functions by providing additional add-on pieces such as, for example, a virtual
15 memory manager.

Also in this integration scenario, an add-on portion of API service 742 that integrates readily with a preexisting API service is provided to support VDE function calls. A pre-existing API
20 service integrated with an add-on portion supports an enhanced set of operating system calls including both calls to VDE functions 604 and calls to functions 606 other than VDE functions (see Figure 11A). The add-on portion of API service

742 may translate VDE function calls into RPC calls for routing by RPC manager 732.

5 ROS 602 may use a standard communications manager 776 provided by the preexisting operating system, or it may provide "add ons" and/or substitutions to it that may be readily integrated into it. Redirector 684 may provide this integration function.

10 This leaves a requirement for ROS 602 to integrate with a preexisting file system 687. Redirector 684 provides this integration function.

15 In this integration scenario, file system 687 of the preexisting operating system is used for all accesses to secondary storage. However, VDE objects 300 may be stored on secondary storage in the form of external object repository 728, file system 687, or remotely accessible through communications manager 776. When object switch 734 wants to access external object repository 728, it makes a request to the object repository manager 770 that then routes the request to object repository 20 728 or to redirector 692 (which in turn accesses the object in file system 687).

Generally, redirector 684 maps VDE object repository 728 content into preexisting calls to file system 687. The redirector 684 provides preexisting OS level information about a VDE object 300, including mapping the object into a preexisting OS's name space. This permits seamless access to VDE protected content using "normal" file system 687 access techniques provided by a preexisting operating system.

In the integration scenarios discussed above, each preexisting target OS file system 687 has different interface requirements by which the redirector mechanism 684 may be "hooked." In general, since all commercially viable operating systems today provide support for network based volumes, file systems, and other devices (e.g., printers, modems, etc.), the redirector 684 may use low level network and file access "hooks" to integrate with a preexisting operating system. "Add-ons" for supporting VDE functions 602 may use these existing hooks to integrate with a preexisting operating system.

20 User Notification Service Manager 740

User Notification Service Manager 740 and associated user notification exception interface ("pop up") 686 provides ROS 602 with an enhanced ability to communicate with a user of electronic appliance 600. Not all applications 608 may be

designed to respond to messaging from ROS 602 passed through API 682, and it may in any event be important or desirable to give ROS 602 the ability to communicate with a user no matter what state an application is in. User notification services
5 manager 740 and interface 686 provides ROS 602 with a mechanism to communicate directly with a user, instead of or in addition to passing a return call through API 682 and an application 608. This is similar, for example, to the ability of the Windows operating system to display a user message in a "dialog
10 box" that displays "on top of" a running application irrespective of the state of the application.

The User Notification 686 block in the preferred embodiment may be implemented as application code. The
15 implementation of interface 740a is preferably built over notification service manager 740, which may be implemented as part of API service manager 742. Notification services manager
740 in the preferred embodiment provides notification support to dispatch specific notifications to an appropriate user process via
20 the appropriate API return, or by another path. This mechanism permits notifications to be routed to any authorized process—not just back to a process that specified a notification mechanism.

API Service Manager 742

The preferred embodiment API Service Manager 742 is implemented as a service interface to the RPC service manager 732. All user API requests are built on top of this basic interface.

5 The API Service Manager 742 preferably provides a service instance for each running user application 608.

Most RPC calls to ROS functions supported by API Service Manager 742 in the preferred embodiment may map directly to
10 service calls with some additional parameter checking. This mechanism permits developers to create their own extended API libraries with additional or changed functionality.

In the scenario discussed above in which ROS 602 is
15 formed by integrating "add ons" with a preexisting operating system, the API service 742 code may be shared (e.g., resident in a host environment like a Windows DLL), or it may be directly linked with an applications's code— depending on an application programmer's implementation decision, and/or the type of
20 electronic appliance 600. The Notification Service Manager 740 may be implemented within API 682. These components interface with Notification Service component 686 to provide a transition between system and user space.

Secure Database Service Manager ("SDSM") 744

There are at least two ways that may be used for managing secure database 600:

- a commercial database approach, and
- 5 • a site record number approach.

Which way is chosen may be based on the number of records that a VDE site stores in the secure database 610.

10 The commercial database approach uses a commercial database to store securely wrapped records in a commercial database. This way may be preferred when there are a large number of records that are stored in the secure database 610. This way provides high speed access, efficient updates, and easy integration to host systems at the cost of resource usage (most
15 commercial database managers use many system resources).

20 The site record number approach uses a "site record number" ("SRN") to locate records in the system. This scheme is preferred when the number of records stored in the secure database 610 is small and is not expected to change extensively over time. This way provides efficient resources use with limited update capabilities. SRNs permit further grouping of similar data records to speed access and increase performance.

Since VDE 100 is highly scalable, different electronic appliances 600 may suggest one way more than the other. For example, in limited environments like a set top, PDA, or other low end electronic appliance, the SRN scheme may be preferred
5 because it limits the amount of resources (memory and processor) required. When VDE is deployed on more capable electronic appliances 600 such as desktop computers, servers and at clearinghouses, the commercial database scheme may be more desirable because it provides high performance in environments
10 where resources are not limited.

One difference between the database records in the two approaches is whether the records are specified using a full VDE ID or SRN. To translate between the two schemes, a SRN
15 reference may be replaced with a VDE ID database reference wherever it occurs. Similarly, VDE IDs that are used as indices or references to other items may be replaced by the appropriate SRN value.

20 In the preferred embodiment, a commercially available database manager 730 is used to maintain secure database 610. ROS 602 interacts with commercial database manager 730 through a database driver 750 and a database interface 748. The database interface 748 between ROS 602 and external, third

party database vendors' commercial database manager 730 may be an open standard to permit any database vendor to implement a VDE compliant database driver 750 for their products.

5 ROS 602 may encrypt each secure database 610 record so that a VDE-provided security layer is "on top of" the commercial database structure. In other words, SPE 736 may write secure records in sizes and formats that may be stored within a database record structure supported by commercial database
10 manager 730. Commercial database manager 730 may then be used to organize, store, and retrieve the records. In some embodiments, it may be desirable to use a proprietary and/or newly created database manager in place of commercial database manager 730. However, the use of commercial database
15 manager 730 may provide certain advantages such as, for example, an ability to use already existing database management product(s).

 The Secure Database Services Manager ("SDSM") 744
20 makes calls to an underlying commercial database manager 730 to obtain, modify, and store records in secure database 610. In the preferred embodiment, "SDSM" 744 provides a layer "on top of" the structure of commercial database manager 730. For example, all VDE-secure information is sent to commercial

database manager 730 in encrypted form. SDSM 744 in
conjunction with cache manager 746 and database interface 748
may provide record management, caching (using cache manager
746), and related services (on top of) commercial database
5 systems 730 and/or record managers. Database Interface 748
and cache manager 746 in the preferred embodiment do not
present their own RSI, but rather the RPC Manager 732
communicates to them through the Secure Database Manager
RSI 744a.

10

Name Services Manager 752

The Name Services Manager 752 supports three
subservices: user name services, host name services, and
services name services. User name services provides mapping
15 and lookup between user name and user ID numbers, and may
also support other aspects of user-based resource and
information security. Host name services provides mapping and
lookup between the names (and other information, such as for
example address, communications connection/routing
20 information, etc.) of other processing resources (e.g., other host
electronic appliances) and VDE node IDs. Services name service
provides a mapping and lookup between services names and
other pertinent information such as connection information (e.g.,

remotely available service routing and contact information) and service IDs.

5 Name Services Manager 752 in the preferred embodiment is connected to External Services Manager 772 so that it may provide external service routing information directly to the external services manager. Name services manager 752 is also connected to secure database manager 744 to permit the name services manager 752 to access name services records stored
10 within secure database 610.

External Services Manager 772 & Services Transport Layer 786

 The External Services Manager 772 provides protocol support capabilities to interface to external service providers.
15 External services manager 772 may, for example, obtain external service routing information from name services manager 752, and then initiate contact to a particular external service (e.g., another VDE electronic appliance 600, a financial clearinghouse, etc.) through communications manager 776. External services
20 manager 772 uses a service transport layer 786 to supply communications protocols and other information necessary to provide communications.

There are several important examples of the use of External Services Manager 772. Some VDE objects may have some or all of their content stored at an Object Repository 728 on an electronic appliance 600 other than the one operated by a user who has, or wishes to obtain, some usage rights to such VDE objects. In this case, External Services Manager 772 may manage a connection to the electronic appliance 600 where the VDE objects desired (or their content) is stored. In addition, file system 687 may be a network file system (e.g., Netware, LANtastic, NFS, etc.) that allows access to VDE objects using redirecter 684. Object switch 734 also supports this capability.

If External Services Manager 772 is used to access VDE objects, many different techniques are possible. For example, the VDE objects may be formatted for use with the World Wide Web protocols (HTML, HTTP, and URL) by including relevant headers, content tags, host ID to URL conversion (e.g., using Name Services Manager 752) and an HTTP-aware instance of Services Transport Layer 786.

In other examples, External Services Manager 772 may be used to locate, connect to, and utilize remote event processing services; smart agent execution services (both to provide these services and locate them); certification services for Public Keys;

remote Name Services; and other remote functions either supported by ROS 602 RPCs (e.g., have RSIs), or using protocols supported by Services Transport Layer 786.

5 **Outgoing Administrative Object Manager 754**

Outgoing administrative object manager 754 receives administrative objects from object switch 734, object repository manager 770 or other source for transmission to another VDE electronic appliance. Outgoing administrative object manager 754 takes care of sending the outgoing object to its proper destination. Outgoing administrative object manager 754 may obtain routing information from name services manager 752, and may use communications service 776 to send the object. Outgoing administrative object manager 754 typically maintains records (in concert with SPE 503) in secure database 610 (e.g., shipping table 444) that reflect when objects have been successfully transmitted, when an object should be transmitted, and other information related to transmission of objects.

20 **Incoming Administrative Object Manager 756**

Incoming administrative object manager 756 receives administrative objects from other VDE electronic appliances 600 via communications manager 776. It may route the object to object repository manager 770, object switch 734 or other

destination. Incoming administrative object manager 756 typically maintains records (in concert with SPE 503) in secure database 610 (e.g., receiving table 446) that record which objects have been received, objects expected for receipt, and other
5 information related to received and/or expected objects.

Object Repository Manager 770

Object repository manager 770 is a form of database or file manager. It manages the storage of VDE objects 300 in object
10 repository 728, in a database, or in the file system 687. Object repository manager 770 may also provide the ability to browse and/or search information related to objects (such as summaries of content, abstracts, reviewers' commentary, schedules, promotional materials, etc.), for example, by using
15 INFORMATION methods associated with VDE objects 300.

Object Submittal Manager 774

Object submittal manager 774 in the preferred embodiment provides an interface between an application 608
20 and object switch 734, and thus may be considered in some respects part of API 682. For example, it may allow a user application to create new VDE objects 300. It may also allow incoming/outgoing administrative object managers 756, 754 to create VDE objects 300 (administrative objects).

Figure 12A shows how object submittal manager 774 may be used to communicate with a user of electronic appliance 600 to help to create a new VDE object 300. Figure 12A shows that object creation may occur in two stages in the preferred
5 embodiment: an object definition stage 1220, and an object creation stage 1230. The role of object submittal manager 774 is indicated by the two different "user input" depictions (774(1), 774(2)) shown in Figure 12A.

10 In one of its roles or instances, object submittal manager 774 provides a user interface 774a that allows the user to create an object configuration file 1240 specifying certain characteristics of a VDE object 300 to be created. This user interface 774a may, for example, allow the user to specify that
15 she wants to create an object, allow the user to designate the content the object will contain, and allow the user to specify certain other aspects of the information to be contained within the object (e.g., rules and control information, identifying information, etc.).

20 Part of the object definition task 1220 in the preferred embodiment may be to analyze the content or other information to be placed within an object. Object definition user interface 774a may issue calls to object switch 734 to analyze "content" or

other information that is to be included within the object to be created in order to define or organize the content into "atomic elements" specified by the user. As explained elsewhere herein, such "atomic element" organizations might, for example, break
5 up the content into paragraphs, pages or other subdivisions specified by the user, and might be explicit (e.g., inserting a control character between each "atomic element") or implicit. Object switch 734 may receive static and dynamic content (e.g., by way of time independent stream interface 762 and real time
10 stream interface 760), and is capable of accessing and retrieving stored content or other information stored within file system 687.

The result of object definition 1240 may be an object configuration file 1240 specifying certain parameters relating to
15 the object to be created. Such parameters may include, for example, map tables, key management specifications, and event method parameters. The object construction stage 1230 may take the object configuration file 1240 and the information or content to be included within the new object as input, construct
20 an object based on these inputs, and store the object within object repository 728.

Object construction stage 1230 may use information in object configuration file 1240 to assemble or modify a container.

5 This process typically involves communicating a series of events to SPE 503 to create one or more PERCs 808, public headers, private headers, and to encrypt content, all for storage in the new object 300 (or within secure database 610 within records associated with the new object).

10 The object configuration file 1240 may be passed to container manager 764 within object switch 734. Container manager 734 is responsible for constructing an object 300 based on the object configuration file 1240 and further user input. The user may interact with the object construction 1230 through another instance 774.2 of object submittal manager 774. In this further user interaction provided by object submittal manager 774, the user may specify permissions, rules and/or control information to be applied to or associated with the new object 15 300. To specify permissions, rules and control information, object submittal manager 774 and/or container manager 764 within object switch 734 generally will, as mentioned above, need to issue calls to SPE 503 (e.g., through gateway 734) to cause the SPE to obtain appropriate information from secure database 610, 20 generate appropriate database items, and store the database items into the secure database 610 and/or provide them in encrypted, protected form to the object switch for incorporation into the object. Such information provided by SPE 503 may

include, in addition to encrypted content or other information,
one or more PERCs 808, one or more method cores 1000', one or
more load modules 1100, one or more data structures such as
UDEs 1200 and/or MDEs 1202, along with various key blocks,
5 tags, public and private headers. and error correction
information.

The container manager 764 may, in cooperation with SPE
503. construct an object container 302 based at least in part on
10 parameters about new object content or other information as
specified by object configuration file 1240. Container manager
764 may then insert into the container 302 the content or other
information (as encrypted by SPE 503) to be included in the new
object. Container manager 764 may also insert appropriate
15 permissions, rules and/or control information into the container
302 (this permissions, rules and/or control information may be
defined at least in part by user interaction through object
submittal manager 774, and may be processed at least in part by
SPE 503 to create secure data control structures). Container
20 manager 764 may then write the new object to object repository
687, and the user or the electronic appliance may "register" the
new object by including appropriate information within secure
database 610.

Communications Subsystem 776

Communications subsystem 776, as discussed above, may be a conventional communications service that provides a network manager 780 and a mail gateway manager 782. Mail filters 784 may be provided to automatically route objects 300 and other VDE information to/from the outside world. Communications subsystem 776 may support a real time content feed 684 from a cable, satellite or other telecommunications link.

Secure Processing Environment 503

As discussed above in connection with Figure 12, each electronic appliance 600 in the preferred embodiment includes one or more SPEs 503 and/or one or more HPEs 655. These secure processing environments each provide a protected execution space for performing tasks in a secure manner. They may fulfill service requests passed to them by ROS 602, and they may themselves generate service requests to be satisfied by other services within ROS 602 or by services provided by another VDE electronic appliance 600 or computer.

20

In the preferred embodiment, an SPE 503 is supported by the hardware resources of an SPU 500. An HPE 655 may be supported by general purpose processor resources and rely on software techniques for security/protection. HPE 655 thus gives

ROS 602 the capability of assembling and executing certain component assemblies 690 on a general purpose CPU such as a microcomputer, minicomputer, mainframe computer or supercomputer processor. In the preferred embodiment, the overall software architecture of an SPE 503 may be the same as the software architecture of an HPE 655. An HPE 655 can "emulate" SPE 503 and associated SPU 500, i.e., each may include services and resources needed to support an identical set of service requests from ROS 602 (although ROS 602 may be restricted from sending to an HPE certain highly secure tasks to be executed only within an SPU 500).

Some electronic appliance 600 configurations might include both an SPE 503 and an HPE 655. For example, the HPE 655 could perform tasks that need lesser (or no) security protections, and the SPE 503 could perform all tasks that require a high degree of security. This ability to provide serial or concurrent processing using multiple SPE and/or HPE arrangements provides additional flexibility, and may overcome limitations imposed by limited resources that can practically or cost-effectively be provided within an SPU 500. The cooperation of an SPE 503 and an HPE 655 may, in a particular application, lead to a more efficient and cost effective but nevertheless secure overall processing environment for supporting and providing the

secure processing required by VDE 100. As one example, an HPE 655 could provide overall processing for allowing a user to manipulate released object 300 'contents,' but use SPE 503 to access the secure object and release the information from the object.

Figure 13 shows the software architecture of the preferred embodiment Secure Processing Environment (SPE) 503. This architecture may also apply to the preferred embodiment Host Processing Environment (HPE) 655. "Protected Processing Environment" ("PPE") 650 may refer generally to SPE 503 and/or HPE 655. Hereinafter, unless context indicates otherwise, references to any of "PPE 650," "HPE 655" and "SPE 503" may refer to each of them.

As shown in Figure 13, SPE 503 (PPE 650) includes the following service managers/major functional blocks in the preferred embodiment:

Kernel/Dispatcher 552

- Channel Services Manager 562
- SPE RPC Manager 550
- Time Base Manager 554
- Encryption/Decryption Manager 556
- Key and Tag Manager 558

- Summary Services Manager 560
- Authentication Manager/Service Communications Manager 564
- Random Value Generator 565
- 5 • Secure Database Manager 566
- Other Services 592.

Each of the major functional blocks of PPE 650 is discussed in detail below.

10

I. SPE Kernel/Dispatcher 552

The Kernel Dispatcher 552 provides an operating system "kernel" that runs on and manages the hardware resources of SPU 500. This operating system "kernel" 552 provides a self-
15 contained operating system for SPU 500; it is also a part of overall ROS 602 (which may include multiple OS kernels, including one for each SPE and HPE ROS is controlling/managing). Kernel/dispatcher 552 provides SPU task and memory management, supports internal SPU hardware
20 interrupts, provides certain "low level services," manages "DTD" data structures, and manages the SPU bus interface unit 530. Kernel/dispatcher 552 also includes a load module execution manager 568 that can load programs into secure execution space for execution by SPU 500.

In the preferred embodiment, kernel/dispatcher 552 may include the following software/functional components:

- load module execution manager 568
- task manager 576
- 5 memory manager 578
- virtual memory manager 580
- "low level" services manager 582
- internal interrupt handlers 584
- BIU handler 586 (may not be present in HPE 655)
- 10 Service interrupt queues 588
- DTD Interpreter 590.

At least parts of the kernel/dispatcher 552 are preferably stored in SPU firmware loaded into SPU ROM 532. An example
15 of a memory map of SPU ROM 532 is shown in Figure 14A. This memory map shows the various components of kernel/dispatcher 552 (as well as the other SPE services shown in Figure 13) residing in SPU ROM 532a and/or EEPROM 532b. The Figure
14B example of an NVRAM 534b memory map shows the task
20 manager 576 and other information loaded into NVRAM.

One of the functions performed by kernel/dispatcher 552 is to receive RPC calls from ROS RPC manager 732. As explained above, the ROS Kernel RPC manager 732 can route RPC calls to

the SPE 503 (via SPE Device Driver 736 and its associated RSI 736a) for action by the SPE. The SPE kernel/dispatcher 552 receives these calls and either handles them or passes them on to SPE RPC manager 550 for routing internally to SPE 503. SPE 503 based processes can also generate RPC requests. Some of these requests can be processed internally by the SPE 503. If they are not internally serviceable, they may be passed out of the SPE 503 through SPE kernel/dispatcher 552 to ROS RPC manager 732 for routing to services external to SPE 503.

10

A. Kernel/Dispatcher Task Management

Kernel dispatcher task manager 576 schedules and oversees tasks executing within SPE 503 (PPE 650). SPE 503 supports many types of tasks. A "channel" (a special type of task that controls execution of component assemblies 690 in the preferred embodiment) is treated by task manager 576 as one type of task. Tasks are submitted to the task manager 576 for execution. Task manager 576 in turn ensures that the SPE 503/SPU 500 resources necessary to execute the tasks are made available, and then arranges for the SPU microprocessor 520 to execute the task.

15

20

Any call to kernel/dispatcher 552 gives the kernel an opportunity to take control of SPE 503 and to change the task or

tasks that are currently executing. Thus, in the preferred embodiment kernel/dispatcher task manager 576 may (in conjunction with virtual memory manager 580 and/or memory manager 578) "swap out" of the execution space any or all of the tasks that are currently active, and "swap in" additional or different tasks.

SPE tasking managed by task manager 576 may be either "single tasking" (meaning that only one task may be active at a time) or "multi-tasking" (meaning that multiple tasks may be active at once). SPE 503 may support single tasking or multi-tasking in the preferred embodiment. For example, "high end" implementations of SPE 503 (e.g., in server devices) should preferably include multi-tasking with "preemptive scheduling." Desktop applications may be able to use a simpler SPE 503, although they may still require concurrent execution of several tasks. Set top applications may be able to use a relatively simple implementation of SPE 503, supporting execution of only one task at a time. For example, a typical set top implementation of SPU 500 may perform simple metering, budgeting and billing using subsets of VDE methods combined into single "aggregate" load modules to permit the various methods to execute in a single tasking environment. However, an execution environment that supports only single tasking may limit use with more

complex control structures. Such single tasking versions of SPE 503 trade flexibility in the number and types of metering and budgeting operations for smaller run time RAM size requirements. Such implementations of SPE 503 may
5 (depending upon memory limitations) also be limited to metering a single object 300 at a time. Of course, variations or combinations are possible to increase capabilities beyond a simple single tasking environment without incurring the additional cost required to support "full multitasking."

10

In the preferred embodiment, each task in SPE 503 is represented by a "swap block," which may be considered a "task" in a traditional multitasking architecture. A "swap block" in the preferred embodiment is a bookkeeping mechanism used by task
15 manager 576 to keep track of tasks and subtasks. It corresponds to a chunk of code and associated references that "fits" within the secure execution environment provided by SPU 500. In the preferred embodiment, it contains a list of references to shared data elements (e.g., load modules 1100 and UDEs 1200), private
20 data elements (e.g., method data and local stack), and swapped process "context" information (e.g., the register set for the process when it is not processing). Figure 14C shows an example of a snapshot of SPU RAM 532 storing several examples of "swap blocks" for a number of different tasks/methods such as a

"channel" task, a "control" task, an "event" task, a "meter" task, a "budget" task, and a "billing" task. Depending on the size of SPU RAM 532, "swap blocks" may be swapped out of RAM and stored temporarily on secondary storage 652 until their execution can
5 be continued. Thus, SPE 503 operating in a multi-tasking mode may have one or more tasks "sleeping." In the simplest form, this involves an active task that is currently processing, and another task (e.g., a control task under which the active task may be running) that is "sleeping" and is "swapped out" of active
10 execution space. Kernel dispatcher 522 may swap out tasks at any time.

Task manager 576 may use Memory Manager 578 to help it perform this swapping operation. Tasks may be swapped out
15 of the secure execution space by reading appropriate information from RAM and other storage internal to SPU 500, for example, and writing a "swap block" to secondary storage 652. Kernel 552 may swap a task back into the secure execution space by reading the swap block from secondary storage 652 and writing the
20 appropriate information back into SPU RAM 532. Because secondary storage 652 is not secure, SPE 503 must encrypt and cryptographically seal (e.g., using a one-way hash function initialized with a secret value known only inside the SPU 500) each swap block before it writes it to secondary storage. The

SPE 503 must decrypt and verify the cryptographic seal for each swap block read from secondary storage 652 before the swap block can be returned to the secure execution space for further execution.

5

Loading a "swap block" into SPU memory may require one or more "paging operations" to possibly first save, and then flush, any "dirty pages" (i.e., pages changed by SPE 503) associated with the previously loaded swap blocks, and to load all required pages for the new swap block context.

10

Kernel dispatcher 522 preferably manages the "swap blocks" using service interrupt queues 588. These service interrupt queues 588 allow kernel/dispatcher 552 to track tasks (swap blocks) and their status (running, "swapped out," or "asleep"). The kernel/dispatcher 552 in the preferred embodiment may maintain the following service interrupt queues 588 to help it manage the "swap blocks":

15

RUN queue

20

SWAP queue

SLEEP queue.

Those tasks that are completely loaded in the execution space and are waiting for and/or using execution cycles from microprocessor 502 are in the RUN queue. Those tasks that are

"swapped" out (e.g., because they are waiting for other swappable components to be loaded) are referenced in the SWAP queue. Those tasks that are "asleep" (e.g., because they are "blocked" on some resource other than processor cycles or are not needed at the moment) are referenced in the SLEEP queue.

5 Kernel/dispatcher task manager 576 may, for example, transition tasks between the RUN and SWAP queues based upon a "round-robin" scheduling algorithm that selects the next task waiting for service, swaps in any pieces that need to be paged in, and executes the task. Kernel/dispatcher 552 task manager 576

10 may transition tasks between the SLEEP queue and the "awake" (i.e., RUN or SWAP) queues as needed.

When two or more tasks try to write to the same data structure in a multi-tasking environment, a situation exists that may result in "deadly embrace" or "task starvation." A "multi-threaded" tasking arrangement may be used to prevent "deadly embrace" or "task starvation" from happening. The preferred embodiment kernel/dispatcher 552 may support "single

15 threaded" or "multi-threaded" tasking.

20

In single threaded applications, the kernel/dispatcher 552 "locks" individual data structures as they are loaded. Once locked, no other SPE 503 task may load them and will "block"

waiting for the data structure to become available. Using a single threaded SPE 503 may, as a practical matter, limit the ability of outside vendors to create load modules 1100 since there can be no assurance that they will not cause a "deadly embrace" with other VDE processes about which outside vendors may know little or nothing. Moreover, the context swapping of a partially updated record might destroy the integrity of the system, permit unmetered use, and/or lead to deadlock. In addition, such "locking" imposes a potentially indeterminate delay into a typically time critical process, may limit SPE 503 throughput, and may increase overhead.

This issue notwithstanding, there are other significant processing issues related to building single-threaded versions of SPE 503 that may limit its usefulness or capabilities under some circumstances. For example, multiple concurrently executing tasks may not be able to process using the same often-needed data structure in a single-threaded SPE 503. This may effectively limit the number of concurrent tasks to one. Additionally, single-threadedness may eliminate the capability of producing accurate summary budgets based on a number of concurrent tasks since multiple concurrent tasks may not be able to effectively share the same summary budget data structure. Single-threadedness may also eliminate the capability to support

audit processing concurrently with other processing. For example, real-time feed processing might have to be shut down in order to audit budgets and meters associated with the monitoring process.

5

One way to provide a more workable "single-threaded" capability is for kernel/dispatcher 552 to use virtual page handling algorithms to track "dirty pages" as data areas are written to. The "dirty pages" can be swapped in and out with the task swap block as part of local data associated with the swap block. When a task exits, the "dirty pages" can be merged with the current data structure (possibly updated by another task for SPU 500) using a three-way merge algorithm (i.e., merging the original data structure, the current data structure, and the "dirty pages" to form a new current data structure). During the update process, the data structure can be locked as the pages are compared and swapped. Even though this virtual paging solution might be workable for allowing single threading in some applications, the vendor limitations mentioned above may limit the use of such single threaded implementations in some cases to dedicated hardware. Any implementation that supports multiple users (e.g., "smart home" set tops, many desk tops and certain PDA applications, etc.) may hit limitations of a single threaded device in certain circumstances.

10

15

20

It is preferable when these limitations are unacceptable to use a full "multi-threaded" data structure write capabilities. For example, a type of "two-phase commit" processing of the type used by database vendors may be used to allow data structure sharing between processes. To implement this "two-phase commit" process, each swap block may contain page addresses for additional memory blocks that will be used to store changed information. A change page is a local copy of a piece of a data element that has been written by an SPE process. The changed page(s) references associated with a specific data structure are stored locally to the swap block in the preferred embodiment.

For example, SPE 503 may support two (change pages) per data structure. This limit is easily alterable by changing the size of the swap block structure and allowing the update algorithm to process all of the changed pages. The "commit" process can be invoked when a swap block that references changed pages is about to be discarded. The commit process takes the original data element that was originally loaded (e.g., UDE_0), the current data element (e.g., UDE_n) and the changed pages, and merges them to create a new copy of the data element (e.g., UDE_{n+1}). Differences can be resolved by the DTD interpreter 590 using a DTD for the data element. The original data element is

discarded (e.g., as determined by its DTD use count) if no other swap block references it.

B. Kernel/Dispatcher Memory Management

5 Memory manager 578 and virtual memory manager 580 in the preferred embodiment manage ROM 532 and RAM 534 memory within SPU 500 in the preferred embodiment. Virtual memory manager 580 provides a fully "virtual" memory system to increase the amount of "virtual" RAM available in the SPE
10 secure execution space beyond the amount of physical RAM 534a provided by SPU 500. Memory manager 578 manages the memory in the secure execution space, controlling how it is accessed, allocated and deallocated. SPU MMU 540, if present, supports virtual memory manager 580 and memory manager 578
15 in the preferred embodiment. In some "minimal" configurations of SPU 500 there may be no virtual memory capability and all memory management functions will be handled by memory manager 578. Memory management can also be used to help enforce the security provided by SPE 503. In some classes of
20 SPUs 500, for example, the kernel memory manager 578 may use hardware memory management unit (MMU) 540 to provide page level protection within the SPU 500. Such a hardware-based memory management system provides an effective

mechanism for protecting VDE component assemblies 690 from compromise by "rogue" load modules.

5 In addition, memory management provided by memory manager 578 operating at least in part based on hardware-based MMU 540 may securely implement and enforce a memory architecture providing multiple protection domains. In such an architecture, memory is divided into a plurality of domains that are largely isolated from each other and share only specific
10 memory areas under the control of the memory manager 578. An executing process cannot access memory outside its domain and can only communicate with other processes through services provided by and mediated by privileged kernel/dispatcher software 552 within the SPU 500. Such an architecture is more
15 secure if it is enforced at least in part by hardware within MMU 540 that cannot be modified by any software-based process executing within SPU 500.

In the preferred embodiment, access to services
20 implemented in the ROM 532 and to physical resources such as NVRAM 534b and RTC 528 are mediated by the combination of privileged kernel/dispatcher software 552 and hardware within MMU 540. ROM 532 and RTC 528 requests are privileged in

order to protect access to critical system component routines
(e.g., RTC 528).

5 Memory manager 578 is responsible for allocating and
deallocating memory; supervising sharing of memory resources
between processes; and enforcing memory access/use restriction.
The SPE kernel/dispatcher memory manager 578 typically
initially allocates all memory to kernel 552, and may be
configured to permit only process-level access to pages as they
10 are loaded by a specific process. In one example SPE operating
system configuration, memory manager 578 allocates memory
using a simplified allocation mechanism. A list of each memory
page accessible in SPE 503 may be represented using a bit map
allocation vector, for example. In a memory block, a group of
15 contiguous memory pages may start at a specific page number.
The size of the block is measured by the number of memory
pages it spans. Memory allocation may be recorded by
setting/clearing the appropriate bits in the allocation vector.

20 To assist in memory management functions, a "dope
vector" may be prepended to a memory block. The "dope vector"
may contain information allowing memory manager 578 to
manage that memory block. In its simplest form, a memory
block may be structured as a "dope vector" followed by the actual

memory area of the block. This "dope vector" may include the block number, support for dynamic paging of data elements, and a marker to detect memory overwrites. Memory manager 578 may track memory blocks by their block number and convert the
5 block number to an address before use. All accesses to the memory area can be automatically offset by the size of the "dope vector" during conversion from a block memory to a physical address. "Dope vectors" can also be used by virtual memory manager 580 to help manage virtual memory.

10

The ROM 532 memory management task performed by memory manager 578 is relatively simple in the preferred embodiment. All ROM 532 pages may be flagged as "read only" and as "non-pagable." EEPROM 532B memory management
15 may be slightly more complex since the "burn count" for each EEPROM page may need to be retained. SPU EEPROM 532B may need to be protected from all uncontrolled writes to conserve the limited writable lifetime of certain types of this memory. Furthermore, EEPROM pages may in some cases not be the
20 same size as memory management address pages.

SPU NVRAM 534b is preferably battery backed RAM that has a few access restrictions. Memory manager 578 can ensure control structures that must be located in NVRAM 534b are not

relocated during "garbage collection" processes. As discussed above, memory manager 578 (and MMU 540 if present) may protect NVRAM 534b and RAM 534a at a page level to prevent tampering by other processes.

5

Virtual memory manager 580 provides paging for programs and data between SPU external memory and SPU internal RAM 534a. It is likely that data structures and executable processes will exceed the limits of any SPU 500 internal memory. For example, PERCs 808 and other fundamental control structures may be fairly large, and "bit map meters" may be, or become, very large. This eventuality may be addressed in two ways:

10

(1) subdividing load modules 1100; and

15

(2) supporting virtual paging.

20

Load modules 1100 can be "subdivided" in that in many instances they can be broken up into separate components only a subset of which must be loaded for execution. Load modules 1100 are the smallest pagable executable element in this example. Such load modules 1100 can be broken up into separate components (e.g., executable code and plural data description blocks), only one of which must be loaded for simple load modules to execute. This structure permits a load module

1100 to initially load only the executable code and to load the data description blocks into the other system pages on a demand basis. Many load modules 1100 that have executable sections that are too large to fit into SPU 500 can be restructured into two or more smaller independent load modules. Large load modules may be manually "split" into multiple load modules that are "chained" together using explicit load module references.

Although "demand paging" can be used to relax some of these restrictions, the preferred embodiment uses virtual paging to manage large data structures and executables. Virtual Memory Manager 580 "swaps" information (e.g., executable code and/or data structures) into and out of SPU RAM 534a, and provides other related virtual memory management services to allow a full virtual memory management capability. Virtual memory management may be important to allow limited resource SPU 500 configurations to execute large and/or multiple tasks.

C. SPE Load Module Execution Manager 568

The SPE (HPE) load module execution manager ("LMEM") 568 loads executables into the memory managed by memory manager 578 and executes them. LMEM 568 provides mechanisms for tracking load modules that are currently loaded inside the protected execution environment. LMEM 568 also

provides access to basic load modules and code fragments stored within, and thus always available to, SPE 503. LMEM 568 may be called, for example, by load modules 1100 that want to execute other load modules.

5

In the preferred embodiment, the load module execution manager 568 includes a load module executor ("program loader") 570, one or more internal load modules 572, and library routines 574. Load module executor 570 loads executables into memory (e.g., after receiving a memory allocation from memory manager 578) for execution. Internal load module library 572 may provide a set of commonly used basic load modules 1100 (stored in ROM 532 or NVRAM 534b, for example). Library routines 574 may provide a set of commonly used code fragments/routines (e.g., bootstrap routines) for execution by SPE 503.

10
15

Library routines 574 may provide a standard set of library functions in ROM 532. A standard list of such library functions along with their entry points and parameters may be used. Load modules 1100 may call these routines (e.g., using an interrupt reserved for this purpose). Library calls may reduce the size of load modules by moving commonly used code into a central location and permitting a higher degree of code reuse. All load modules 1100 for use by SPE 503 are preferably referenced by a

20

load module execution manager 568 that maintains and scans a list of available load modules and selects the appropriate load module for execution. If the load module is not present within SPE 503, the task is "slept" and LMEM 568 may request that the
5 load module 1100 be loaded from secondary storage 562. This request may be in the form of an RPC call to secure database manager 566 to retrieve the load module and associated data structures, and a call to encrypt/decrypt manager 556 to decrypt the load module before storing it in memory allocated by memory
10 manager 578.

In somewhat more detail, the preferred embodiment executes a load module 1100 by passing the load module execution manager 568 the name (e.g., VDE ID) of the desired
15 load module 1100. LMEM 568 first searches the list of "in memory" and "built-in" load modules 572. If it cannot find the desired load module 1100 in the list, it requests a copy from the secure database 610 by issuing an RPC request that may be handled by ROS secure database manager 744 shown in Figure
20 12. Load module execution manager 568 may then request memory manager 578 to allocate a memory page to store the load module 1100. The load module execution manager 568 may copy the load module into that memory page, and queue the page for decryption and security checks by encrypt/decrypt manager 556

and key and tag manager 558. Once the page is decrypted and checked, the load module execution manager 568 checks the validation tag and inserts the load module into the list of paged in modules and returns the page address to the caller. The caller
5 may then call the load module 1100 directly or allow the load module execution module 570 to make the call for it.

Figure 15a shows a detailed example of a possible format for a channel header 596 and a channel 594 containing channel
10 detail records 594(1), 594(2), . . . 594(N). Channel header 596 may include a channel ID field 597(1), a user ID field 597(2), an object ID field 597(3), a field containing a reference or other identification to a "right" (i.e., a collection of events supported by methods referenced in a PERC 808 and/or "user rights table"
15 464) 597(4), an event queue 597(5), and one or more fields 598 that cross-reference particular event codes with channel detail records ("CDRs"). Channel header 596 may also include a "jump" or reference table 599 that permits addressing of elements within an associated component assembly or assemblies 690.
20 Each CDR 594(1), . . . 594(N) corresponds to a specific event (event code) to which channel 594 may respond. In the preferred embodiment, these CDRs may include explicitly and/or by reference each method core 1000' (or fragment thereof), load module 1100 and data structure(s), (e.g., URT, UDE 1200 and/or

MDE 1202) needed to process the corresponding event. In the preferred embodiment, one or more of the CDRs (e.g., 594(1)) may reference a control method and a URT 464 as a data structure.

5

Figure 15b shows an example of program control steps performed by SPE 503 to "open" a channel 594 in the preferred embodiment. In the preferred embodiment, a channel 594 provides event processing for a particular VDE object 300, a particular authorized user, and a particular "right" (i.e., type of event). These three parameters may be passed to SPE 503. Part of SPE kernel/dispatcher 552 executing within a "channel 0" constructed by low level services 582 during a "bootstrap" routine may respond initially to this "open channel" event by allocating an available channel supported by the processing resources of SPE 503 (block 1125). This "channel 0" "open channel" task may then issue a series of requests to secure database manager 566 to obtain the "blueprint" for constructing one or more component assemblies 690 to be associated with channel 594 (block 1127). In the preferred embodiment, this "blueprint" may comprise a PERC 808 and/or URT 464. It may be obtained by using the "Object, User, Right" parameters passed to the "open channel" routine to "chain" together object registration table 460 records, user/object table 462 records, URT 464 records, and PERC 808

10

15

20

records. This "open channel" task may preferably place calls to key and tag manager 558 to validate and correlate the tags associated with these various records to ensure that they are authentic and match. The preferred embodiment process then
5 may write appropriate information to channel header 596 (block 1129). Such information may include, for example, User ID, Object ID, and a reference to the "right" that the channel will process. The preferred embodiment process may next use the "blueprint" to access (e.g., the secure database manager 566
10 and/or from load module execution manager library(ies) 568) the appropriate "control method" that may be used to, in effect, supervise execution of all of the other methods 1000 within the channel 594 (block 1131). The process may next "bind" this control method to the channel (block 1133), which step may
15 include binding information from a URT 464 into the channel as a data structure for the control method. The process may then pass an "initialization" event into channel 594 (block 1135). This "initialization" event may be created by the channel services manager 562, the process that issued the original call requesting
20 a service being fulfilled by the channel being built, or the control method just bound to the channel could itself possibly generate an initialization event which it would in effect pass to itself.

In response to this "initialization" event, the control method may construct the channel detail records 594(1), . . . 594(N) used to handle further events other than the "initialization" event. The control method executing "within" the channel may access the various components it needs to construct associated component assemblies 690 based on the "blueprint" accessed at step 1127 (block 1137). Each of these components is bound to the channel 594 (block 1139) by constructing an associated channel detail record specifying the method core(s) 1000', load module(s) 1100, and associated data structure(s) (e.g., UDE(s) 1200 and/or MDE(s) 1202) needed to respond to the event. The number of channel detail records will depend on the number of events that can be serviced by the "right," as specified by the "blueprint" (i.e., URT 464). During this process, the control method will construct "swap blocks" to, in effect, set up all required tasks and obtain necessary memory allocations from kernel 562. The control method will, as necessary, issue calls to secure database manager 566 to retrieve necessary components from secure database 610, issue calls to encrypt/decrypt manager 556 to decrypt retrieved encrypted information, and issue calls to key and tag manager 558 to ensure that all retrieved components are validated. Each of the various component assemblies 690 so constructed are "bound" to the channel through the channel header event code/pointer records 598 and by constructing

appropriate swap blocks referenced by channel detail records
594(1), . . . 594(N). When this process is complete, the channel
594 has been completely constructed and is ready to respond to
further events. As a last step, the Figure 15b process may, if
5 desired, deallocate the "initialization" event task in order to free
up resources.

Once a channel 594 has been constructed in this fashion, it
will respond to events as they arrive. Channel services manager
10 562 is responsible for dispatching events to channel 594. Each
time a new event arrives (e.g., via an RPC call), channel services
manager 562 examines the event to determine whether a
channel already exists that is capable of processing it. If a
channel does exist, then the channel services manager 562
15 passes the event to that channel. To process the event, it may be
necessary for task manager 576 to "swap in" certain "swappable
blocks" defined by the channel detail records as active tasks. In
this way, executable component assemblies 690 formed during
the channel open process shown in Figure 15b are placed into
20 active secure execution space, the particular component
assembly that is activated being selected in response to the
received event code. The activated task will then perform its
desired function in response to the event.

To destroy a channel, the various swap blocks defined by the channel detail records are destroyed, the identification information in the channel header 596 is wiped clean, and the channel is made available for re-allocation by the "channel 0" "open channel" task.

D. SPE Interrupt Handlers 584

As shown in Figure 13, kernel/dispatcher 552 also provides internal interrupt handler(s) 584. These help to manage the resources of SPU 500. SPU 500 preferably executes in either "interrupt" or "polling" mode for all significant components. In polling mode, kernel/dispatcher 552 may poll each of the sections/circuits within SPU 500 and emulate an interrupt for them. The following interrupts are preferably supported by SPU 500 in the preferred embodiment:

- "tick" of RTC 528
- interrupt from bus interface 530
- power fail interrupt
- watchdog timer interrupt
- interrupt from encrypt/decrypt engine 522
- memory interrupt (e.g., from MMU 540).

When an interrupt occurs, an interrupt controller within microprocessor 520 may cause the microprocessor to begin

executing an appropriate interrupt handler. An interrupt handler is a piece of software/firmware provided by kernel/dispatcher 552 that allows microprocessor 520 to perform particular functions upon the occurrence of an interrupt. The
5 interrupts may be "vectored" so that different interrupt sources may effectively cause different interrupt handlers to be executed.

A "timer tick" interrupt is generated when the real-time RTC 528 "pulses." The timer tick interrupt is processed by a
10 timer tick interrupt handler to calculate internal device date/time and to generate timer events for channel processing.

The bus interface unit 530 may generate a series of interrupts. In the preferred embodiment, bus interface 530,
15 modeled after a USART, generates interrupts for various conditions (e.g., "receive buffer full," "transmitter buffer empty," and "status word change"). Kernel/dispatcher 552 services the transmitter buffer empty interrupt by sending the next character from the transmit queue to the bus interface 530.
20 Kernel/dispatcher interrupt handler 584 may service the received buffer full interrupt by reading a character, appending it to the current buffer, and processing the buffer based on the state of the service engine for the bus interface 530. Kernel/dispatcher 552 preferably processes a status word change

interrupt and addresses the appropriate send/receive buffers accordingly.

5 SPU 500 generates a power fail interrupt when it detects
an imminent power fail condition. This may require immediate
action to prevent loss of information. For example, in the
preferred embodiment, a power fail interrupt moves all recently
written information (i.e., "dirty pages") into non-volatile NVRAM
534b, marks all swap blocks as "swapped out," and sets the
10 appropriate power fail flag to facilitate recovery processing.
Kernel/dispatcher 552 may then periodically poll the "power fail
bit" in a status word until the data is cleared or the power is
removed completely.

15 SPU 500 in the example includes a conventional watchdog
timer that generates watchdog timer interrupts on a regular
basis. A watchdog timer interrupt handler performs internal
device checks to ensure that tampering is not occurring. The
internal clocks of the watchdog timer and RTC 528 are compared
20 to ensure SPU 500 is not being paused or probed, and other
internal checks on the operation of SPU 500 are made to detect
tampering.

The encryption/decryption engine 522 generates an interrupt when a block of data has been processed. The kernel interrupt handler 584 adjusts the processing status of the block being encrypted or decrypted, and passes the block to the next stage of processing. The next block scheduled for the encryption service then has its key moved into the encrypt/decrypt engine 522, and the next cryptographic process started.

A memory management unit 540 interrupt is generated when a task attempts to access memory outside the areas assigned to it. A memory management interrupt handler traps the request, and takes the necessary action (e.g., by initiating a control transfer to memory manager 578 and/or virtual memory manager 580). Generally, the task will be failed, a page fault exception will be generated, or appropriate virtual memory page(s) will be paged in.

E. Kernel/Dispatcher Low Level Services 582

Low level services 582 in the preferred embodiment provide "low level" functions. These functions in the preferred embodiment may include, for example, power-on initialization, device POST, and failure recovery routines. Low level services 582 may also in the preferred embodiment provide (either by themselves or in combination with authentication

manager/service communications manager 564) download
response-challenge and authentication communication protocols,
and may provide for certain low level management of SPU 500
memory devices such as EEPROM and FLASH memory (either
5 alone or in combination with memory manager 578 and/or
virtual memory manager 580).

F. Kernel/Dispatcher BIU handler 586

BIU handler 586 in the preferred embodiment manages
10 the bus interface unit 530 (if present). It may, for example,
maintain read and write buffers for the BIU 530, provide BIU
startup initialization, etc.

G. Kernel/Dispatcher DTD Interpreter 590

15 DTD interpreter 590 in the preferred embodiment handles
data formatting issues. For example, the DTD interpreter 590
may automatically open data structures such as UDEs 1200
based on formatting instructions contained within DTDs.

20 The SPE kernel/dispatcher 552 discussed above supports
all of the other services provided by SPE 503. Those other
services are discussed below.

II. SPU Channel Services Manager 562

"Channels" are the basic task processing mechanism of SPE 503 (HPE 655) in the preferred embodiment. ROS 602 provides an event-driven interface for "methods." A "channel" allows component assemblies 690 to service events. A "channel" is a conduit for passing "events" from services supported by SPE 503 (HPE 655) to the various methods and load modules that have been specified to process these events, and also supports the assembly of component assemblies 690 and interaction between component assemblies. In more detail, "channel" 594 is a data structure maintained by channel manager 593 that "binds" together one or more load modules 1100 and data structures (e.g., UDEs 1200 and/or MDEs 1202) into a component assembly 690. Channel services manager 562 causes load module execution manager 569 to load the component assembly 690 for execution, and may also be responsible for passing events into the channel 594 for response by a component assembly 690. In the preferred embodiment, event processing is handled as a message to the channel service manager 562.

20

Figure 15 is a diagram showing how the preferred embodiment channel services manager 562 constructs a "channel" 594, and also shows the relationship between the channel and component assemblies 690. Briefly, the SPE

channel manager 562 establishes a "channel" 594 and an associated "channel header" 596. The channel 594 and its header 596 comprise a data structure that "binds" or references elements of one or more component assemblies 690. Thus, the
5 channel 594 is the mechanism in the preferred embodiment that collects together or assembles the elements shown in Figure 11E into a component assembly 690 that may be used for event processing.

10 The channel 594 is set up by the channel services manager 562 in response to the occurrence of an event. Once the channel is created, the channel services manager 562 may issue function calls to load module execution manager 568 based on the channel 594. The load module execution manager 568 loads the load
15 modules 1100 referenced by a channel 594, and requests execution services by the kernel/dispatcher task manager 576. The kernel/dispatcher 552 treats the event processing request as a task, and executes it by executing the code within the load modules 1100 referenced by the channel.

20

The channel services manager 562 may be passed an identification of the event (e.g., the "event code"). The channel services manager 562 parses one or more method cores 1000' that are part of the component assembly(ies) 690 the channel

services manager is to assemble. It performs this parsing to determine which method(s) and data structure(s) are invoked by the type of event. Channel manager 562 then issues calls (e.g., to secure database manager 566) to obtain the methods and data structure(s) needed to build the component assembly 690. These called-for method(s) and data structure(s) (e.g., load modules 1100, UDEs 1200 and/or MDEs 1202) are each decrypted using encrypt/decrypt manager 556 (if necessary), and are then each validated using key and tag manager 558. Channel manager 562 constructs any necessary "jump table" references to, in effect, "link" or "bind" the elements into a single cohesive executable so the load module(s) can reference data structures and any other load module(s) in the component assembly. Channel manager 562 may then issue calls to LMEM 568 to load the executable as an active task.

Figure 15 shows that a channel 594 may reference another channel. An arbitrary number of channels 594 may be created by channel manager 594 to interact with one another.

"Channel header" 596 in the preferred embodiment is (or references) the data structure(s) and associated control program(s) that queues events from channel event sources, processes these events, and releases the appropriate tasks

specified in the "channel detail record" for processing. A "channel detail record" in the preferred embodiment links an event to a "swap block" (i.e., task) associated with that event. The "swap block" may reference one or more load modules 1100, UDEs 1200 and private data areas required to properly process the event. One swap block and a corresponding channel detail item is created for each different event the channel can respond to.

10 In the preferred embodiment, Channel Services Manager 562 may support the following (internal) calls to support the creation and maintenance of channels 562:

Call Name	Source	Description
15 Write Event	Write	Writes an event to the channel for response by the channel. The <u>Write Event</u> call thus permit the caller to insert an event into the event queue associated with the channel. The event will be processed in turn by the channel 594.

"Bind Item"	Ioctl	Binds an item to a channel with the appropriate processing algorithm. The <u>Bind Item</u> call permits the caller to bind a VDE item ID to a channel (e.g., to create one or more swap blocks associated with a channel). This call may manipulate the contents of individual swap blocks.
"Unbind Item"	Ioctl	Unbinds an item from a channel with the appropriate processing algorithm. The <u>Unbind Item</u> call permits the caller to break the binding of an item to a swap block. This call may manipulate the contents of individual swap blocks.

5 **SPE RPC Manager 550**

As described in connection with Figure 12, the architecture of ROS 602 is based on remote procedure calls in the preferred embodiment. ROS 602 includes an RPC Manager 732 that passes RPC calls between services each of which present an RPC service interface ("RSI") to the RPC manager. In the preferred embodiment, SPE 503 (HPE 655) is also built around the same RPC concept. The SPE 503 (HPE 655) may include a number of internal modular service providers each presenting an RSI to an RPC manager 550 internal to the SPE (HPE). These internal

service providers may communicate with each other and/or with ROS RPC manager 732 (and thus, with any other service provided by ROS 602 and with external services), using RPC service requests.

5

RPC manager 550 within SPE 503 (HPE 655) is not the same as RPC manager 732 shown in Figure 12, but it performs a similar function within the SPE (HPE): it receives RPC requests and passes them to the RSI presented by the service that is to fulfill the request. In the preferred embodiment, requests are passed between ROS RPC manager 732 and the outside world (i.e., SPE device driver 736) via the SPE (HPE)

10

Kernel/Dispatcher 552. Kernel/Dispatcher 552 may be able to service certain RPC requests itself, but in general it passes

15

received requests to RPC manager 550 for routing to the appropriate service internal to the SPE (HPE). In an alternate embodiment, requests may be passed directly between the HPE, SPE, API, Notification interface, and other external services instead of routing them through the ROS RPC manager 732.

20

The decision on which embodiment to use is part of the scalability of the system; some embodiments are more efficient than others under various traffic loads and system configurations. Responses by the services (and additional service requests they may themselves generate) are provided to RPC

Manager 550 for routing to other service(s) internal or external to SPE 503 (HPE 655).

5 SPE RPC Manager 550 and its integrated service manager
uses two tables to dispatch remote procedure calls: an RPC
services table, and an optional RPC dispatch table. The RPC
services table describes where requests for specific services are to
be routed for processing. In the preferred embodiment, this table
is constructed in SPU RAM 534a or NVRAM 534b, and lists each
10 RPC service "registered" within SPU 500. Each row of the RPC
services table contains a service ID, its location and address, and
a control byte. In simple implementations, the control byte
indicates only that the service is provided internally or
externally. In more complex implementations, the control byte
15 can indicate an instance of the service (e.g., each service may
have multiple "instances" in a multi-tasking environment). ROS
RPC manager 732 and SPE 503 may have symmetric copies of
the RPC services table in the preferred embodiment. If an RPC
service is not found in the RPC services table, SPE 503 may
20 either reject it or pass it to ROS RPC manager 732 for service.

The SPE RPC manager 550 accepts the request from the
RPC service table and processes that request in accordance with
the internal priorities associated with the specific service. In

SPE 503, the RPC service table is extended by an RPC dispatch table. The preferred embodiment RPC dispatch table is organized as a list of Load Module references for each RPC service supported internally by SPE 503. Each row in the table contains a load module ID that services the call, a control byte that indicates whether the call can be made from an external caller, and whether the load module needed to service the call is permanently resident in SPU 500. The RPC dispatch table may be constructed in SPU ROM 532 (or EEPROM) when SPU firmware 508 is loaded into the SPU 500. If the RPC dispatch table is in EEPROM, it flexibly allows for updates to the services without load module location and version control issues.

In the preferred embodiment, SPE RPC manager 550 first references a service request against the RPC service table to determine the location of the service manager that may service the request. The RPC manager 550 then routes the service request to the appropriate service manager for action. Service requests are handled by the service manager within the SPE 503 using the RPC dispatch table to dispatch the request. Once the RPC manager 550 locates the service reference in the RPC dispatch table, the load module that services the request is called and loaded using the load module execution manager 568. The load module execution manager 568 passes control to the

requested load module after performing all required context configuration, or if necessary may first issue a request to load it from the external management files 610.

5 **SPU Time Base Manager 554**

The time base manager 554 supports calls that relate to the real time clock ("RTC") 528. In the preferred embodiment, the time base manager 554 is always loaded and ready to respond to time based requests.

10

The table below lists examples of basic calls that may be supported by the time base manager 554:

15

Call Name	Description
Independent requests	
Get Time	Returns the time (local, GMT, or ticks).
Set time	Sets the time in the RTC 528. Access to this command may be restricted to a VDE administrator.
Adjust time	Changes the time in the RTC 528. Access to this command may be restricted to a VDE administrator.
Set Time Parameter	Set GMT / local time conversion and the current and allowable magnitude of user adjustments to RTC 528 time.
Channel Services Manager Requests	

20

Call Name	Description
Bind Time	Bind timer services to a channel as an event source.
Unbind Time	Unbind timer services from a channel as an event source.
Set Alarm	Sets an alarm notification for a specific time. The user will be notified by an alarm event at the time of the alarm. Parameters to this request determine the event, frequency, and requested processing for the alarm.
Clear Alarm	Cancels a requested alarm notification.

5

SPU Encryption/Decryption Manager 556

The Encryption/Decryption Manager 556 supports calls to the various encryption/decryption techniques supported by SPE 503/HPE 655. It may be supported by a hardware-based encryption/decryption engine 522 within SPU 500. Those encryption/decryption technologies not supported by SPU encrypt/decrypt engine 522 may be provided by encrypt/decrypt manager 556 in software. The primary bulk encryption/decryption load modules preferably are loaded at all times, and the load modules necessary for other algorithms are preferably paged in as needed. Thus, if the primary bulk encryption/decryption algorithm is DES, only the DES load modules need be permanently resident in the RAM 534a of SPE 503/HPE 655.

20

The following are examples of RPC calls supported by
Encrypt/Decrypt Manager 556 in the preferred embodiment:

5	Call Name	Description
	PK Encrypt	Encrypt a block using a PK (public key) algorithm.
	PK Decrypt	Decrypt a block using a PK algorithm.
	DES Encrypt	Encrypt a block using DES.
10	DES Decrypt	Decrypt a block using DES.
	RC-4 Encrypt	Encrypt a block using the RC-4 (or other bulk encryption) algorithm.
	RC-4 Decrypt	Decrypt a block using the RC-4 (or other bulk encryption) algorithm.
15	Initialize DES Instance	Initialize DES instance to be used.
	Initialize RC-4 Instance	Initialize RC-4 instance to be used.
20	Initialize MD5 Instance	Initialize MD5 instance to be used.
25	Process MD5 Block	Process MD5 block.

The call parameters passed may include the key to be
used; mode (encryption or decryption); any needed Initialization

Vectors; the desired cryptographic operating (e.g., type of feedback); the identification of the cryptographic instance to be used; and the start address, destination address, and length of the block to be encrypted or decrypted.

5

SPU Key and Tag Manager 558

The SPU Key and Tag Manager 558 supports calls for key storage, key and management file tag look up, key convolution, and the generation of random keys, tags, and transaction numbers.

10

The following table shows an example of a list of SPE/HPE key and tag manager service 558 calls:

15

20

25

Call Name	Description
Key Requests	
Get Key	Retrieve the requested key.
Set Key	Set (store) the specified key.
Generate Key	Generate a key (pair) for a specified algorithm.
Generate Convolved Key	Generate a key using a specified convolution algorithm and algorithm parameter block.
Get Convolution Algorithm	Return the currently set (default) convolution parameters for a specific convolution algorithm.
Set Convolution Algorithm	Sets the convolution parameters for a specific convolution algorithm (calling routine must provide a tag to read returned contents).
Tag Requests	
Get Tag	Get the validation (or other) tag for a specific VDE Item ID.
Set Tag	Set the validation (or other) tag for a specific VDE Item ID to a known value.

Calculate Hash Block Number	Calculate the "hash block number" for a specific VDE Item ID.
Set Hash Parameters	Set the hash parameters and hash algorithm. Forces a resynchronization of the hash table.
Get Hash Parameters	Retrieve the current hash parameters/algorithm.
Synchronize Management Files	Synchronize the management files and rebuild the hash block tables based on information found in the tables. Reserved for VDE administrator

5
 10
 15
 Keys and tags may be securely generated within SPE 503 (HPE 655) in the preferred embodiment. The key generation algorithm is typically specific to each type of encryption supported. The generated keys may be checked for cryptographic weakness before they are used. A request for Key and Tag Manager 558 to generate a key, tag and/or transaction number preferably takes a length as its input parameter. It generates a random number (or other appropriate key value) of the requested length as its output.

20
 The key and tag manager 558 may support calls to retrieve specific keys from the key storage areas in SPU 500 and any keys stored external to the SPU. The basic format of the calls is to request keys by key type and key number. Many of the keys are periodically updated through contact with the VDE administrator, and are kept within SPU 500 in NVRAM 534b or

EEPROM because these memories are secure, updatable and non-volatile.

5 SPE 503/HPE 655 may support both Public Key type keys and Bulk Encryption type keys. The public key (PK) encryption type keys stored by SPU 500 and managed by key and tag manager 558 may include, for example, a device public key, a device private key, a PK certificate, and a public key for the certificate. Generally, public keys and certificates can be stored
10 externally in non-secured memory if desired, but the device private key and the public key for the certificate should only be stored internally in an SPU 500 EEPROM or NVRAM 534b. Some of the types of bulk encryption keys used by the SPU 500 may include, for example, general-purpose bulk encryption keys,
15 administrative object private header keys, stationary object private header keys, traveling object private header keys, download/initialization keys, backup keys, trail keys, and management file keys.

20 As discussed above, preferred embodiment Key and Tag Manager 558 supports requests to adjust or convolute keys to make new keys that are produced in a deterministic way dependent on site and/or time, for example. Key convolution is an algorithmic process that acts on a key and some set of input

parameter(s) to yield a new key. It can be used, for example, to increase the number of keys available for use without incurring additional key storage space. It may also be used, for example, as a process to "age" keys by incorporating the value of real-time
5 RTC 528 as parameters. It can be used to make keys site specific by incorporating aspects of the site ID as parameters.

Key and Tag Manager 558 also provides services relating to tag generation and management. In the preferred
10 embodiment, transaction and access tags are preferably stored by SPE 503 (HPE 655) in protected memory (e.g., within the NVRAM 534b of SPU 500). These tags may be generated by key and tag manager 558. They are used to, for example, check access rights to, validate and correlate data elements. For
15 example, they may be used to ensure components of the secured data structures are not tampered with outside of the SPU 500. Key and tag manager 558 may also support a trail transaction tag and a communications transaction tag.

20 **SPU Summary Services Manager 560**

SPE 503 maintains an audit trail in reprogrammable non-volatile memory within the SPU 500 and/or in secure database 610. This audit trail may consist of an audit summary of budget activity for financial purposes, and a security summary of SPU

use. When a request is made to the SPU, it logs the request as having occurred and then notes whether the request succeeded or failed. All successful requests may be summed and stored by type in the SPU 500. Failure information, including the
5 elements listed below, may be saved along with details of the failure:

10

Control Information Retained in an SPE on Access Failures	
Object ID	
User ID	
Type of failure	
Time of failure	

15

This information may be analyzed to detect cracking attempts or to determine patterns of usage outside expected (and budgeted) norms. The audit trail histories in the SPU 500 may be retained
20 until the audit is reported to the appropriate parties. This will allow both legitimate failure analysis and attempts to cryptoanalyze the SPU to be noted.

Summary services manager 560 may store and maintain
25 this internal summary audit information. This audit information can be used to check for security breaches or other aspects of the operation of SPE 503. The event summaries may

be maintained, analyzed and used by SPE 503 (HPE 655) or a VDE administrator to determine and potentially limit abuse of electronic appliance 600. In the preferred embodiment, such parameters may be stored in secure memory (e.g., within the
5 NVRAM 534b of SPU 500).

There are two basic structures for which summary services are used in the preferred embodiment. One (the "event summary data structure") is VDE administrator specific and keeps track of
10 events. The event summary structure may be maintained and audited during periodic contact with VDE administrators. The other is used by VDE administrators and/or distributors for overall budget. A VDE administrator may register for event summaries and an overall budget summary at the time an
15 electronic appliance 600 is initialized. The overall budget summary may be reported to and used by a VDE administrator in determining distribution of consumed budget (for example) in the case of corruption of secure management files 610.

Participants that receive appropriate permissions can register
20 their processes (e.g., specific budgets) with summary services manager 560, which may then reserve protected memory space (e.g., within NVRAM 534b) and keep desired use and/or access parameters. Access to and modification of each summary can be controlled by its own access tag.

The following table shows an example of a list of PPE
summary service manager 560 service calls:

5

Call Name	Description
Create summary info	Create a summary service if the user has a "ticket" that permits her to request this service.
Get value	Return the current value of the summary service. The caller must present an appropriate tag (and/or "ticket") to use this request.
Set value	Set the value of a summary service.
Increment	Increment the specified summary service(e.g., a scalar meter summary data area). The caller must present an appropriate tag (and/or "ticket") to use this request.
Destroy	Destroy the specified summary service if the user has a tag and/or "ticket" that permits them to request this service.

10

15

In the preferred embodiment, the event summary data structure uses a fixed event number to index into a look up table. The look up table contains a value that can be configured as a counter or a counter plus limit. Counter mode may be used by VDE administrators to determine device usage. The limit mode may be used to limit tampering and attempts to misuse the electronic appliance 600. Exceeding a limit will result in SPE

503 (HPE 655) refusing to service user requests until it is reset by a VDE administrator. Calls to the system wide event summary process may preferably be built into all load modules that process the events that are of interest.

5

The following table shows examples of events that may be separately metered by the preferred embodiment event summary data structure:

10

Event Type	
Successful Events	Initialization completed successfully.
	User authentication accepted.
	Communications established.
	Channel loads set for specified values.
	Decryption completed.
	Key information updated.
	New budget created or existing budget updated.
	New billing information generated or existing billing updated.
	New meter set up or existing meter updated.
	New PERC created or existing PERC updated.
	New objects registered.
	Administrative objects successfully processed.
	Audit processed successfully.

	All other events.
Failed Events	Initialization failed.
	Authentication failed.
	Communication attempt failed.
	Request to load a channel failed.
	Validation attempt unsuccessful.
	Link to subsidiary item failed correlation tag match.
	Authorization attempt failed.
	Decryption attempt failed.
	Available budget insufficient to complete requested procedure.
	Audit did not occur.
	Administrative object did not process correctly.
	Other failed events.

Another, "overall currency budget" summary data

5 structure maintained by the preferred embodiment summary services manager 560 allows registration of VDE electronic appliance 600. The first entry is used for an overall currency budget consumed value, and is registered by the VDE administrator that first initializes SPE 503 (HPE 655). Certain

10 currency consuming load modules and audit load modules that complete the auditing process for consumed currency budget may call the summary services manager 560 to update the currency consumed value. Special authorized load modules may have

access to the overall currency summary, while additional summaries can be registered for by individual providers.

SPE Authentication Manager/Service Communications Manager 564

The Authentication Manager/Service Communications Manager 564 supports calls for user password validation and "ticket" generation and validation. It may also support secure communications between SPE 503 and an external node or device (e.g., a VDE administrator or distributor). It may support the following examples of authentication-related service requests in the preferred embodiment:

Call Name	Description
User Services	
Create User	Creates a new user and stores Name Services Records (NSRs) for use by the Name Services Manager 752.
Authenticate User	Authenticates a user for use of the system. This request lets the caller authenticate as a specific user ID. Group membership is also authenticated by this request. The authentication returns a "ticket" for the user.
Delete User	Deletes a user's NSR and related records.
Ticket Services	
Generate Ticket	Generates a "ticket" for use of one or more services.

Authenticate Ticket	Authenticates a "ticket."
------------------------	---------------------------

5 Not included in the table above are calls to the secure
communications service. The secure communications service
provided by manager 564 may provide (e.g., in conjunction with
low-level services manager 582 if desired) secure
communications based on a public key (or others) challenge-
10 response protocol. This protocol is discussed in further detail
elsewhere in this document. Tickets identify users with respect
to the electronic appliance 600 in the case where the appliance
may be used by multiple users. Tickets may be requested by and
returned to VDE software applications through a ticket-granting
15 protocol (e.g., Kerberos). VDE components may require tickets to
be presented in order to authorize particular services.

SPE Secure Database Manager 566

20 Secure database manager 566 retrieves, maintains and
stores secure database records within secure database 610 on
memory external to SPE 503. Many of these secure database
files 610 are in encrypted form. All secure information retrieved
by secure database manager 566 therefore must be decrypted by
encrypt/decrypt manager 556 before use. Secure information
25 (e.g., records of use) produced by SPE 503 (HPE 655) which must

be stored external to the secure execution environment are also encrypted by encrypt/decrypt manager 556 before they are stored via secure database manager 566 in a secure database file 610.

5 For each VDE item loaded into SPE 503, Secure Database manager 566 in the preferred embodiment may search a master list for the VDE item ID, and then check the corresponding transaction tag against the one in the item to ensure that the item provided is the current item. Secure Database Manager
10 566 may maintain list of VDE item ID and transaction tags in a "hash structure" that can be paged into SPE 503 to quickly locate the appropriate VDE item ID. In smaller systems, a look up table approach may be used. In either case, the list should be structured as a pagable structure that allows VDE item ID to be
15 located quickly.

 The "hash based" approach may be used to sort the list into "hash buckets" that may then be accessed to provide more rapid and efficient location of items in the list. In the "hash
20 based" approach, the VDE item IDs are "hashed" through a subset of the full item ID and organized as pages of the "hashed" table. Each "hashed" page may contain the rest of the VDE item ID and current transaction tag for each item associated with that page. The "hash" table page number may be derived from the

components of the VDE item ID, such as distribution ID, item ID, site ID, user ID, transaction tag, creator ID, type and/or version. The hashing algorithm (both the algorithm itself and the parameters to be hashed) may be configurable by a VDE administrator on a site by site basis to provide optimum hash page use. An example of a hash page structure appears below:

	Field
10	Hash Page Header
	Distributor ID
	Item ID
	Site ID
	User ID
15	Transaction Tag
	Hash Page Entry
	Creator ID
	Item ID
	Type
20	Version
	Transaction Tag

In this example, each hash page may contain all of the VDE item IDs and transaction tags for items that have identical distributor ID, item ID, and user ID fields (site ID will be fixed for a given electronic appliance 600). These four pieces of information may thus be used as hash algorithm parameters.

The "hash" pages may themselves be frequently updated, and should carry transaction tags that are checked each time a "hash" page is loaded. The transaction tag may also be updated each time a "hash" page is written out.

5

As an alternative to the hash-based approach, if the number of updatable items is kept small (such as in a dedicated consumer electronic appliance 600), then assigning each updatable item a unique sequential site record number as part of its VDE item ID may allow a look up table approach to be used. Only a small number of bytes of transaction tag are needed per item, and a table transaction tag for all frequently updatable items can be kept in protected memory such as SPU NVRAM 534b.

10

Random Value Generator Manager 565

Random Value Generator Manager 565 may generate random values. If a hardware-based SPU random value generator 542 is present, the Random Value Generator Manager 565 may use it to assist in generating random values.

15

Other SPE RPC Services 592

Other authorized RPC services may be included in SPU 500 by having them "register" themselves in the RPC Services

Table and adding their entries to the RPC Dispatch Table. For example, one or more component assemblies 690 may be used to provide additional services as an integral part of SPE 503 and its associated operating system. Requests to services not registered in these tables will be passed out of SPE 503 (HPE 655) for external servicing.

SPE 503 Performance Considerations

Performance of SPE 503 (HPE 655) is a function of:

- complexity of the component assemblies used
- number of simultaneous component assembly operations
- amount of internal SPU memory available
- speed of algorithm for block encryption/decryption

The complexity of component assembly processes along with the number of simultaneous component assembly processes is perhaps the primary factor in determining performance. These factors combine to determine the amount of code and data and must be resident in SPU 500 at any one time (the minimum device size) and thus the number of device size "chunks" the processes must be broken down into. Segmentation inherently increases run time size over simpler models. Of course, feature limited versions of SPU 500 may be implemented using significantly smaller amounts of RAM 534. "Aggregate" load

modules as described above may remove flexibility in configuring VDE structures and also further limit the ability of participants to individually update otherwise separated elements, but may result in a smaller minimum device size. A very simple metering
5 version of SPU 500 can be constructed to operate with minimal device resources.

The amount of RAM 534 internal to SPU 500 has more impact on the performance of the SPE 503 than perhaps any
10 other aspect of the SPU. The flexible nature of VDE processes allows use of a large number of load modules, methods and user data elements. It is impractical to store more than a small number of these items in ROM 532 within SPU 500. Most of the code and data structures needed to support a specific VDE
15 process will need to be dynamically loaded into the SPU 500 for the specific VDE process when the process is invoked. The operating system within SPU 500 then may page in the necessary VDE items to perform the process. The amount of RAM 534 within SPU 500 will directly determine how large any
20 single VDE load module plus its required data can be, as well as the number of page swaps that will be necessary to run a VDE process. The SPU I/O speed, encryption/decryption speed, and the amount of internal memory 532, 534 will directly affect the number of page swaps required in the device. Insecure external

memory may reduce the wait time for swapped pages to be loaded into SPU 500, but will still incur substantial encryption/decryption penalty for each page.

5 In order to maintain security, SPE 503 must encrypt and cryptographically seal each block being swapped out to a storage device external to a supporting SPU 500, and must similarly decrypt, verify the cryptographic seal for, and validate each block as it is swapped into SPU 500. Thus, the data movement and
10 encryption/decryption overhead for each swap block has a very large impact on SPE performance.

 The performance of an SPU microprocessor 520 may not significantly impact the performance of the SPE 503 it supports
15 if the processor is not responsible for moving data through the encrypt/decrypt engine 522.

VDE Secure Database 610

 VDE 100 stores separately deliverable VDE elements in a
20 secure (e.g., encrypted) database 610 distributed to each VDE electronic appliance 610. The database 610 in the preferred embodiment may store and/or manage three basic classes of VDE items:

 VDE objects,

VDE process elements, and
VDE data structures.

The following table lists examples of some of the VDE
5 items stored in or managed by information stored in secure
database 610:

Class		Brief Description
Objects	Content Objects	Provide a container for content.
	Administrative Objects	Provide a container for information used to keep VDE 100 operating.
	Traveling Objects	Provide a container for content and control information.
	Smart Objects	Provide a container for (user-specified) processes and data.
Process Elements	Method Cores	Provide a mechanism to relate events to control mechanisms and permissions.
	Load Modules ("LMs")	Secure (tamper-resistant) executable code.
	Method Data Elements ("MDEs")	Independently deliverable data structures used to control/customize methods.
Data Structures	Permissions Records ("PERCs")	Permissions to use objects; "blueprints" to build component assemblies.

Class		Brief Description
	User Data Elements ("UDEs")	Basic data structure for storing information used in conjunction with load modules.
	Administrative Data Structures	Used by VDE node to maintain administrative information.

Each electronic appliance 600 may have an instance of a secure database 610 that securely maintains the VDE items.

5 Figure 16 shows one example of a secure database 610. The secure database 610 shown in this example includes the following VDE-protected items:

- one or more PERCs 808;
- methods 1000 (including static and dynamic method
- 10 "cores" 1000, and MDEs 1202);
- Static UDEs 1200a and Dynamic UDEs 1200b; and
- load modules 1100.

15 Secure database 610 may also include the following additional data structures used and maintained for administrative purposes:

- an "object registry" 450 that references an object storage 728 containing one or more VDE objects;
- name service records 452; and

- configuration records 454 (including site configuration records 456 and user configuration records 458).

5 Secure database 610 in the preferred embodiment does not include VDE objects 300, but rather references VDE objects stored, for example, on file system 687 and/or in a separate object repository 728. Nevertheless, an appropriate "starting point" for understanding VDE-protected information may be a discussion
10 of VDE objects 300.

VDE Objects 300

VDE 100 provides a media independent container model for encapsulating content. Figure 17 shows an example of a
15 "logical" structure or format 800 for an object 300 provided by the preferred embodiment.

The generalized "logical object" structure 800 shown in Figure 17 used by the preferred embodiment supports digital
20 content delivery over any currently used media. "Logical object" in the preferred embodiment may refer collectively to: content; computer software and/or methods used to manipulate, record, and/or otherwise control use of said content; and permissions, limitations, administrative control information and/or

requirements applicable to said content, and/or said computer software and/or methods. Logical objects may or may not be stored, and may or may not be present in, or accessible to, any given electronic appliance 600. The content portion of a logical
5 object may be organized as information contained in, not contained in, or partially contained in one or more objects.

Briefly, the Figure 17 "logical object" structure 800 in the preferred embodiment includes a public header 802, private
10 header 804, a "private body" 806 containing one or more methods 1000, permissions record(s) (PERC) 808 (which may include one or more key blocks 810), and one or more data blocks or areas 812. These elements may be "packaged" within a "container" 302. This generalized, logical object structure 800 is used in the
15 preferred embodiment for different types of VDE objects 300 categorized by the type and location of their content.

The "container" concept is a convenient metaphor used to give a name to the collection of elements required to make use of
20 content or to perform an administrative-type activity. Container 302 typically includes identifying information, control structures and content (e.g., a property or administrative data). The term "container" is often (e.g., Bento/OpenDoc and OLE) used to describe a collection of information stored on a computer

system's secondary storage system(s) or accessible to a computer system over a communications network on a "server's" secondary storage system. The "container" 302 provided by the preferred embodiment is not so limited or restricted. In VDE 100, there is
5 no requirement that this information is stored together, received at the same time, updated at the same time, used for only a single object, or be owned by the same entity. Rather, in VDE 100 the container concept is extended and generalized to include real-time content and/or online interactive content passed to an
10 electronic appliance over a cable, by broadcast, or communicated by other electronic communication means.

Thus, the "complete" VDE container 302 or logical object structure 800 may not exist at the user's location (or any other
15 location, for that matter) at any one time. The "logical object" may exist over a particular period of time (or periods of time), rather than all at once. This concept includes the notion of a "virtual container" where important container elements may exist either as a plurality of locations and/or over a sequence of
20 time periods (which may or may not overlap). Of course, VDE 100 containers can also be stored with all required control structures and content together. This represents a continuum: from all content and control structures present in a single

container, to no locally accessible content or container specific control structures.

5 Although at least some of the data representing the object is typically encrypted and thus its structure is not discernible, within a PPE 650 the object may be viewed logically as a "container" 302 because its structure and components are automatically and transparently decrypted.

10 A container model merges well with the event-driven processes and ROS 602 provided by the preferred embodiment. Under this model, content is easily subdivided into small, easily manageable pieces, but is stored so that it maintains the structural richness inherent in unencrypted content. An object
15 oriented container model (such as Bento/OpenDoc or OLE) also provides many of the necessary "hooks" for inserting the necessary operating system integration components, and for defining the various content specific methods.

20 In more detail, the logical object structure 800 provided by the preferred embodiment includes a public (or unencrypted) header 802 that identifies the object and may also identify one or more owners of rights in the object and/or one or more distributors of the object. Private (or encrypted) header 804 may

include a part or all of the information in the public header and further, in the preferred embodiment, will include additional data for validating and identifying the object 300 when a user attempts to register as a user of the object with a service clearinghouse, VDE administrator, or an SPU 500.

Alternatively, information identifying one or more rights owners and/or distributors of the object may be located in encrypted form within encrypted header 804, along with any of said additional validating and identifying data.

Each logical object structure 800 may also include a "private body" 806 containing or referencing a set of methods 1000 (i.e., programs or procedures) that control use and distribution of the object 300. The ability to optionally incorporate different methods 1000 with each object is important to making VDE 100 highly configurable. Methods 1000 perform the basic function of defining what users (including, where appropriate, distributors, client administrators, etc.), can and cannot do with an object 300. Thus, one object 300 may come with relatively simple methods, such as allowing unlimited viewing within a fixed period of time for a fixed fee (such as the newsstand price of a newspaper for viewing the newspaper for a period of one week after the paper's publication), while other

objects may be controlled by much more complicated (e.g., billing and usage limitation) methods.

5 Logical object structure 800 shown in Figure 17 may also include one or more PERCs 808. PERCs 808 govern the use of an object 300, specifying methods or combinations of methods that must be used to access or otherwise use the object or its contents. The permission records 808 for an object may include key block(s) 810, which may store decryption keys for accessing the
10 content of the encrypted content stored within the object 300.

The content portion of the object is typically divided into portions called data blocks 812. Data blocks 812 may contain any sort of electronic information, such as, "content," including
15 computer programs, images, sound, VDE administrative information, etc. The size and number of data blocks 812 may be selected by the creator of the property. Data blocks 812 need not all be the same size (size may be influenced by content usage, database format, operating system, security and/or other
20 considerations). Security will be enhanced by using at least one key block 810 for each data block 812 in the object, although this is not required. Key blocks 810 may also span portions of a plurality of data blocks 812 in a consistent or pseudo-random manner. The spanning may provide additional security by

applying one or more keys to fragmented or seemingly random pieces of content contained in an object 300, database, or other information entity.

5 Many objects 300 that are distributed by physical media and/or by "out of channel" means (e.g., redistributed after receipt by a customer to another customer) might not include key blocks 810 in the same object 300 that is used to transport the content protected by the key blocks. This is because VDE objects may
10 contain data that can be electronically copied outside the confines of a VDE node. If the content is encrypted, the copies will also be encrypted and the copier cannot gain access to the content unless she has the appropriate decryption key(s). For
15 objects in which maintaining security is particularly important, the permission records 808 and key blocks 810 will frequently be distributed electronically, using secure communications techniques (discussed below) that are controlled by the VDE
20 nodes of the sender and receiver. As a result, permission records 808 and key blocks 810 will frequently, in the preferred embodiment, be stored only on electronic appliances 600 of registered users (and may themselves be delivered to the user as part of a registration/initialization process). In this instance, permission records 808 and key blocks 810 for each property can be encrypted with a private DES key that is stored only in the

secure memory of an SPU 500, making the key blocks unusable on any other user's VDE node. Alternately, the key blocks 810 can be encrypted with the end user's public key, making those key blocks usable only to the SPU 500 that stores the
5 corresponding private key (or other, acceptably secure, encryption/security techniques can be employed).

In the preferred embodiment, the one or more keys used to encrypt each permission record 808 or other management
10 information record will be changed every time the record is updated (or after a certain one or more events). In this event, the updated record is re-encrypted with new one or more keys. Alternately, one or more of the keys used to encrypt and decrypt management information may be "time aged" keys that
15 automatically become invalid after a period of time. Combinations of time aged and other event triggered keys may also be desirable; for example keys may change after a certain number of accesses, and/or after a certain duration of time or absolute point in time. The techniques may also be used
20 together for any given key or combination of keys. The preferred embodiment procedure for constructing time aged keys is a one-way convolution algorithm with input parameters including user and site information as well as a specified portion of the real time value provided by the SPU RTC 528. Other techniques for

time aging may also be used, including for example techniques that use only user or site information, absolute points in time, and/or duration of time related to a subset of activities related to using or decrypting VDE secured content or the use of the VDE system.

VDE 100 supports many different types of "objects" 300 having the logical object structure 800 shown in Figure 17. Objects may be classified in one sense based on whether the protection information is bound together with the protected information. For example, a container that is bound by its control(s) to a specific VDE node is called a "stationary object" (see Figure 18). A container that is not bound by its control information to a specific VDE node but rather carries sufficient control and permissions to permit its use, in whole or in part, at any of several sites is called a "Traveling Object" (see Figure 19).

Objects may be classified in another sense based on the nature of the information they contain. A container with information content is called a "Content Object" (see Figure 20). A container that contains transaction information, audit trails, VDE structures, and/or other VDE control/administrative information is called an "Administrative Object" (see Figure 21). Some containers that contain executable code operating under

VDE control (as opposed to being VDE control information) are called "Smart Objects." Smart Objects support user agents and provide control for their execution at remote sites. There are other categories of objects based upon the location, type and access mechanism associated with their content, that can include combinations of the types mentioned above. Some of these objects supported by VDE 100 are described below. Some or all of the data blocks 812 shown in Figure 17 may include "embedded" content, administrative, stationary, traveling and/or other objects.

1. Stationary Objects

Figure 18 shows an example of a "Stationary Object" structure 850 provided by the preferred embodiment.

"Stationary Object" structure 850 is intended to be used only at specific VDE electronic appliance/installations that have received explicit permissions to use one or more portions of the stationary object. Therefore, stationary object structure 850 does not contain a permissions record (PERC) 808; rather, this permissions record is supplied and/or delivered separately (e.g., at a different time, over a different path, and/or by a different party) to the appliance/installation 600. A common PERC 808 may be used with many different stationary objects.

As shown in Figure 18, public header 802 is preferably "plaintext" (i.e., unencrypted). Private header 804 is preferably encrypted using at least one of many "private header keys." Private header 804 preferably also includes a copy of

5 identification elements from public header 802, so that if the identification information in the plaintext public header is tampered with, the system can determine precisely what the tamperer attempted to alter. Methods 1000 may be contained in a section called the "private body" 806 in the form of object local

10 methods, load modules, and/or user data elements. This private body (method) section 806 is preferably encrypted using one or more private body keys contained in the separate permissions record 808. The data blocks 812 contain content (information or administrative) that may be encrypted using one or more content

15 keys also provided in permissions record 808.

2. Traveling Objects

Figure 19 shows an example of a "traveling object" structure 860 provided by the preferred embodiment. Traveling

20 objects are objects that carry with them sufficient information to enable at least some use of at least a portion of their content when they arrive at a VDE node.

Traveling object structure 860 may be the same as stationary object structure 850 shown in Figure 18 except that the traveling object structure includes a permissions record (PERC) 808 within private header 804. The inclusion of PERC 808 within traveling object structure 860 permits the traveling object to be used at any VDE electronic appliance/participant 600 (in accordance with the methods 1000 and the contained PERC 808).

10 "Traveling" objects are a class of VDE objects 300 that can specifically support "out of channel" distribution. Therefore, they include key blocks 810 and are transportable from one electronic appliance 600 to another. Traveling objects may come with a quite limited usage related budget so that a user may use, 15 in whole or part, content (such as a computer program, game, or database) and evaluate whether to acquire a license or further license or purchase object content. Alternatively, traveling object PERCs 808 may contain or reference budget records with, for example:

- 20 (a) budget(s) reflecting previously purchased rights or credit for future licensing or purchasing and enabling at least one or more types of object content usage, and/or

- (b) budget(s) that employ (and may debit) available credit(s) stored on and managed by the local VDE node in order to enable object content use, and/or
- 5 (c) budget(s) reflecting one or more maximum usage criteria before a report to a local VDE node (and, optionally, also a report to a clearinghouse) is required and which may be followed by a reset allowing further usage, and/or modification of one or
- 10 more of the original one or more budget(s).

As with standard VDE objects 300, a user may be required to contact a clearinghouse service to acquire additional budgets if the user wishes to continue to use the traveling object after the

15 exhaustion of an available budget(s) or if the traveling object (or a copy thereof) is moved to a different electronic appliance and the new appliance does not have a available credit budget(s) that corresponds to the requirements stipulated by permissions record 808.

20

For example, a traveling object PERC 808 may include a reference to a required budget VDE 1200 or budget options that may be found and/or are expected to be available. For example, the budget VDE may reference a consumer's VISA, MC, AMEX,

or other "generic" budget that may be object independent and may be applied towards the use of a certain or classes of traveling object content (for example any movie object from a class of traveling objects that might be Blockbuster Video rentals). The budget VDE itself may stipulate one or more classes of objects it may be used with, while an object may specifically reference a certain one or more generic budgets. Under such circumstances, VDE providers will typically make information available in such a manner as to allow correct referencing and to enable billing handling and resulting payments.

Traveling objects can be used at a receiving VDE node electronic appliance 600 so long as either the appliance carries the correct budget or budget type (e.g. sufficient credit available from a clearinghouse such as a VISA budget) either in general or for specific one or more users or user classes, or so long as the traveling object itself carries with it sufficient budget allowance or an appropriate authorization (e.g., a stipulation that the traveling object may be used on certain one or more installations or installation classes or users or user classes where classes correspond to a specific subset of installations or users who are represented by a predefined class identifiers stored in a secure database 610). After receiving a traveling object, if the user

(and/or installation) doesn't have the appropriate budget(s) and/or authorizations, then the user could be informed by the electronic appliance 600 (using information stored in the traveling object) as to which one or more parties the user could
5 contact. The party or parties might constitute a list of alternative clearinghouse providers for the traveling object from which the user selects his desired contact).

As mentioned above, traveling objects enable objects 300 to
10 be distributed "Out-Of-Channel;" that is, the object may be distributed by an unauthorized or not explicitly authorized individual to another individual. "Out of channel" includes paths of distribution that allow, for example, a user to directly redistribute an object to another individual. For example, an
15 object provider might allow users to redistribute copies of an object to their friends and associates (for example by physical delivery of storage media or by delivery over a computer network) such that if a friend or associate satisfies any certain criteria required for use of said object, he may do so.

20

For example, if a software program was distributed as a traveling object, a user of the program who wished to supply it or a usable copy of it to a friend would normally be free to do so. Traveling Objects have great potential commercial significance,

since useful content could be primarily distributed by users and through bulletin boards, which would require little or no distribution overhead apart from registration with the "original" content provider and/or clearinghouse.

5

The "out of channel" distribution may also allow the provider to receive payment for usage and/or otherwise maintain at least a degree of control over the redistributed object. Such certain criteria might involve, for example, the registered presence at a user's VDE node of an authorized third party financial relationship, such as a credit card, along with sufficient available credit for said usage.

10

Thus, if the user had a VDE node, the user might be able to use the traveling object if he had an appropriate, available budget available on his VDE node (and if necessary, allocated to him), and/or if he or his VDE node belonged to a specially authorized group of users or installations and/or if the traveling object carried its own budget(s).

15

20

Since the content of the traveling object is encrypted, it can be used only under authorized circumstances unless the traveling object private header key used with the object is broken—a potentially easier task with a traveling object as

compared to, for example, permissions and/or budget information since many objects may share the same key, giving a cryptanalyst both more information in cyphertext to analyze and a greater incentive to perform cryptanalysis.

5

In the case of a "traveling object," content owners may distribute information with some or all of the key blocks 810 included in the object 300 in which the content is encapsulated. Putting keys in distributed objects 300 increases the exposure to attempts to defeat security mechanisms by breaking or cryptanalyzing the encryption algorithm with which the private header is protected (e.g., by determining the key for the header's encryption). This breaking of security would normally require considerable skill and time, but if broken, the algorithm and key could be published so as to allow large numbers of individuals who possess objects that are protected with the same key(s) and algorithm(s) to illegally use protected information. As a result, placing keys in distributed objects 300 may be limited to content that is either "time sensitive" (has reduced value after the passage of a certain period of time), or which is somewhat limited in value, or where the commercial value of placing keys in objects (for example convenience to end-users, lower cost of eliminating the telecommunication or other means for delivering keys and/or permissions information and/or the ability to

10

15

20

supporting objects going "out-of-channel") exceeds the cost of vulnerability to sophisticated hackers. As mentioned elsewhere, the security of keys may be improved by employing convolution techniques to avoid storing "true" keys in a traveling object, although in most cases using a shared secret provided to most or all VDE nodes by a VDE administrator as an input rather than site ID and/or time in order to allow objects to remain independent of these values.

As shown in Figure 19 and discussed above, a traveling object contains a permissions record 808 that preferably provides at least some budget (one, the other, or both, in a general case). Permission records 808 can, as discussed above, contain a key block(s) 810 storing important key information. PERC 808 may also contain or refer to budgets containing potentially valuable quantities/values. Such budgets may be stored within a traveling object itself, or they may be delivered separately and protected by highly secure communications keys and administrative object keys and management database techniques.

The methods 1000 contained by a traveling object will typically include an installation procedure for "self registering" the object using the permission records 808 in the object (e.g., a

REGISTER method). This may be especially useful for objects that have time limited value, objects (or properties) for which the end user is either not charged or is charged only a nominal fee (e.g., objects for which advertisers and/or information publishers are charged based on the number of end users who actually access published information), and objects that require widely available budgets and may particularly benefit from out-of-channel distribution (e.g., credit card derived budgets for objects containing properties such as movies, software programs, games, etc.). Such traveling objects may be supplied with or without contained budget UDEs.

One use of traveling objects is the publishing of software, where the contained permission record(s) may allow potential customers to use the software in a demonstration mode, and possibly to use the full program features for a limited time before having to pay a license fee, or before having to pay more than an initial trial fee. For example, using a time based billing method and budget records with a small pre-installed time budget to allow full use of the program for a short period of time. Various control methods may be used to avoid misuse of object contents. For example, by setting the minimum registration interval for the traveling object to an appropriately large period of time (e.g.,

a month, or six months or a year), users are prevented from re-using the budget records in the same traveling object.

Another method for controlling the use of traveling objects is to include time-aged keys in the permission records that are incorporated in the traveling object. This is useful generally for traveling objects to ensure that they will not be used beyond a certain date without re-registration, and is particularly useful for traveling objects that are electronically distributed by broadcast, network, or telecommunications (including both one and two way cable), since the date and time of delivery of such traveling objects aging keys can be set to accurately correspond to the time the user came into possession of the object.

Traveling objects can also be used to facilitate "moving" an object from one electronic appliance 600 to another. A user could move a traveling object, with its incorporated one or more permission records 808 from a desktop computer, for example, to his notebook computer. A traveling object might register its user within itself and thereafter only be useable by that one user. A traveling object might maintain separate budget information, one for the basic distribution budget record, and another for the "active" distribution budget record of the registered user. In this

way, the object could be copied and passed to another potential user, and then could be a portable object for that user.

5 Traveling objects can come in a container which contains other objects. For example, a traveling object container can include one or more content objects and one or more administrative objects for registering the content object(s) in an end user's object registry and/or for providing mechanisms for enforcing permissions and/or other security functions. Contained
10 administrative objects may be used to install necessary permission records and or budget information in the end user's electronic appliance.

Content Objects

15 Figure 20 shows an example of a VDE content object structure 880. Generally, content objects 880 include or provide information content. This "content" may be any sort of electronic information. For example, content may include: computer software, movies, books, music, information databases,
20 multimedia information, virtual reality information, machine instructions, computer data files, communications messages and/or signals, and other information, at least a portion of which is used and/or manipulated by one or more electronic appliances. VDE 100 can also be configured for authenticating, controlling,

and/or auditing electronic commercial transactions and communications such as inter-bank transactions, electronic purchasing communications, and the transmission of, auditing of, and secure commercial archiving of, electronically signed contracts and other legal documents; the information used for these transactions may also be termed "content." As mentioned above, the content need not be physically stored within the object container but may instead be provided separately at a different time (e.g., a real time feed over a cable).

Content object structure 880 in the particular example shown in Figure 20 is a type of stationary object because it does not include a PERC 308. In this example, content object structure 880 includes, as at least part of its content 812, at least one embedded content object 882 as shown in Figure 5A. Content object structure 880 may also include an administrative object 870. Thus, objects provided by the preferred embodiment may include one or more "embedded" objects.

Administrative Objects

Figure 21 shows an example of an administrative object structure 870 provided by the preferred embodiment. An "administrative object" generally contains permissions, administrative control information, computer software and/or

methods associated with the operation of VDE 100.

Administrative objects may also or alternatively contain records of use, and/or other information used in, or related to, the operation of VDE 100. An administrative object may be distinguished from a content object by the absence of VDE protected "content" for release to an end user for example. Since objects may contain other objects, it is possible for a single object to contain one or more content containing objects and one or more administrative objects. Administrative objects may be used to transmit information between electronic appliances for update, usage reporting, billing and/or control purposes. They contain information that helps to administer VDE 100 and keep it operating properly. Administrative objects generally are sent between two VDE nodes, for example, a VDE clearinghouse service, distributor, or client administrator and an end user's electronic appliance 600.

Administrative object structure 870 in this example includes a public header 802, private header 804 (including a "PERC" 808) and a "private body" 806 containing methods 1000. Administrative object structure 870 in this particular example shown in Figure 20 is a type of traveling object because it contains a PERC 808, but the administrative object could exclude the PERC 808 and be a stationary object. Rather than storing

information content. administrative object structure 870 stores
"administrative information content" 872. Administrative
information content 872 may, for example, comprise a number of
records 872a, 872b, . . . 872n each corresponding to a different
5 "event." Each record 872a, 872b, . . . 872n may include an
"event" field 874, and may optionally include a parameter field
876 and/or a data field 878. These administrative content
records 872 may be used by VDE 100 to define events that may
be processed during the course of transactions, e.g., an event
10 designed to add a record to a secure database might include
parameters 896 indicating how and where the record should be
stored and data field 878 containing the record to be added. In
another example, a collection of events may describe a financial
transaction between the creator(s) of an administrative object
15 and the recipient(s), such as a purchase, a purchase order, or an
invoice. Each event record 872 may be a set of instructions to be
executed by the end user's electronic appliance 600 to make an
addition or modification to the end user's secure database 610,
for example. Events can perform many basic management
20 functions, for example: add an object to the object registry,
including providing the associated user/group record(s), rights
records, permission record and/or method records; delete audit
records (by "rolling up" the audit trail information into, for
example, a more condensed, e.g. summary form, or by actual

deletion); add or update permissions records 808 for previously registered objects; add or update budget records; add or update user rights records; and add or update load modules.

5 In the preferred embodiment, an administrative object may be sent, for example, by a distributor, client administrator, or, perhaps, a clearinghouse or other financial service provider, to an end user, or, alternatively, for example, by an object creator to a distributor or service clearinghouse. Administrative objects,
10 for example, may increase or otherwise adjust budgets and/or permissions of the receiving VDE node to which the administrative object is being sent. Similarly, administrative objects containing audit information in the data area 878 of an event record 872 can be sent from end users to distributors,
15 and/or clearinghouses and/or client administrators, who might themselves further transmit to object creators or to other participants in the object's chain of handling.

Methods

20 Methods 1000 in the preferred embodiment support many of the operations that a user encounters in using objects and communicating with a distributor. They may also specify what method fields are displayable to a user (e.g., use events, user request events, user response events, and user display events).

Additionally, if distribution capabilities are supported in the method, then the method may support distribution activities, distributor communications with a user about a method, method modification, what method fields are displayable to a distributor, and any distribution database checks and record keeping (e.g., distribution events, distributor request events, and distributor response events).

Given the generality of the existing method structure, and the diverse array of possibilities for assembling methods, a generalized structure may be used for establishing relationships between methods. Since methods 1000 may be independent of an object that requires them during any given session, it is not possible to define the relationships within the methods themselves. "Control methods" are used in the preferred embodiment to define relationships between methods. Control methods may be object specific, and may accommodate an individual object's requirements during each session.

A control method of an object establishes relationships between other methods. These relationships are parameterized with explicit method identifiers when a record set reflecting desired method options for each required method is constructed during a registration process.

An "aggregate method" in the preferred embodiment represents a collection of methods that may be treated as a single unit. A collection of methods that are related to a specific property, for example, may be stored in an aggregate method.

5 This type of aggregation is useful from an implementation point of view because it may reduce bookkeeping overhead and may improve overall database efficiency. In other cases, methods may be aggregated because they are logically coupled. For example, two budgets may be linked together because one of the

10 budgets represents an overall limitation, and a second budget represents the current limitation available for use. This would arise if, for example, a large budget is released in small amounts over time.

15 For example, an aggregate method that includes meter, billing and budget processes can be used instead of three separate methods. Such an aggregate method may reference a single "load module" 1100 that performs all of the functions of the three separate load modules and use only one user data

20 element that contains meter, billing and budget data. Using an aggregate method instead of three separate methods may minimize overall memory requirements, database searches, decryptions, and the number of user data element writes back to a secure database 610. The disadvantage of using an aggregate

method instead of three separate methods can be a loss of some flexibility on the part of a provider and user in that various functions may no longer be independently replaceable.

5 Figure 16 shows methods 1000 as being part of secure database 610.

10 A "method" 1000 provided by the preferred embodiment is a collection of basic instructions and information related to the basic instructions, that provides context, data, requirements and/or relationships for use in performing, and/or preparing to perform, the basic instructions in relation to the operation of one or more electronic appliances 600. As shown in Figure 16, methods 1000 in the preferred embodiment are represented in
15 secure database 610 by:

- method "cores" 1000';
- Method Data Elements (MDEs) 1202;
- User Data Elements (UDEs) 1200; and
- Data Description Elements (DTDs).

20

Method "core" 1000' in the preferred embodiment may contain or reference one or more data elements such as MDEs 1202 and UDEs 1200. In the preferred embodiment, MDEs 1202 and UDEs 1200 may have the same general characteristics, the

main difference between these two types of data elements being that a UDE is preferably tied to a particular method as well as a particular user or group of users, whereas an MDE may be tied to a particular method but may be user independent. These

5 MDE and UDE data structures 1200, 1202 are used in the preferred embodiment to provide input data to methods 1000, to receive data outputted by methods, or both. MDEs 1202 and UDEs 1200 may be delivered independently of method cores 1000' that reference them, or the data structures may be

10 delivered as part of the method cores. For example, the method core 1000' in the preferred embodiment may contain one or more MDEs 1202 and/or UDEs 1200 (or portions thereof). Method core 1000' may, alternately or in addition, reference one or more MDE and/or UDE data structures that are delivered

15 independently of method core(s) that reference them.

Method cores 1000' in the preferred embodiment also reference one or more "load modules" 1100. Load modules 1100 in the preferred embodiment comprise executable code, and may

20 also include or reference one or more data structures called "data descriptor" ("DTD") information. This "data descriptor" information may, for example, provide data input information to the DTD interpreter 590. DTDs may enable load modules 1100

to access (e.g., read from and/or write to) the MDE and/or UDE data elements 1202, 1200.

5 Method cores 1000' may also reference one or more DTD and/or MDE data structures that contain a textual description of their operations suitable for inclusion as part of an electronic contract. The references to the DTD and MDE data structures may occur in the private header of the method core 1000', or may be specified as part of the event table described below.

10

Figure 22 shows an example of a format for a method core 1000' provided by the preferred embodiment. A method core 1000' in the preferred embodiment contains a method event table 1006 and a method local data area 1008. Method event
15 table 1006 lists "events." These "events" each reference "load modules" 1100 and/or PERCs 808 that control processing of an event. Associated with each event in the list is any static data necessary to parameterize the load module 1000 or permissions record 808, and reference(s) into method user data area 1008
20 that are needed to support that event. The data that parameterizes the load module 1100 can be thought of, in part, as a specific function call to the load module, and the data elements corresponding to it may be thought of as the input and/or output data for that specific function call.

Method cores 1000' can be specific to a single user, or they may be shared across a number of users (e.g., depending upon the uniqueness of the method core and/or the specific user data element). Specifically, each user/group may have its own UDE
5 1200 and use a shared method core 1000'. This structure allows for lower database overhead than when associating an entire method core 1000' with a user/group. To enable a user to use a method, the user may be sent a method core 1000' specifying a UDE 1200. If that method core 1000' already exists in the site's
10 secure database 610, only the UDE 1200 may need to be added. Alternately, the method may create any required UDE 1200 at registration time.

The Figure 22 example of a format for a method core 1000'
15 provided by the preferred embodiment includes a public (unencrypted) header 802, a private (encrypted) header 804, method event table 1006, and a method local data area 1008.

An example of a possible field layout for method core 1000'
20 public header 802 is shown in the following table:

Field Type		Description
Method ID	Creator ID	Site ID of creator of this method.
	Distributor ID	Distributor of this method (e.g., last change).
	Type ID	Constant, indicates method "type."
	Method ID	Unique sequence number for this method.
	Version ID	Version number of this method.
Other classification information	Class ID	ID to support different method "classes."
	Type ID	ID to support method type compatible searching.
Descriptive Information	Description(s)	Textual description(s) of the method.
	Event Summary	Summary of event classes (e.g., USE) that this method supports.

5

10

An example of a possible field layout for private header 804 is shown below:

Field Type		Description
Copy of Public Header 802 Method ID and "Other Classification Information"		Method ID from Public Header
Descriptive Information	# of Events	# of events supported in this method.
Access and Reference Tags	Access tag	Tags used to determine if this method is the correct method under management by the SPU; ensure that the method core 1000' is used only under appropriate circumstances.
	Validation tag	
	Correlation tag	
Data Structure Reference		Optional Reference to DTD(s) and/or MDE(s)
Check Value		Check value for Private Header and method event table.
Check Value for Public Header		Check Value for Public Header

Referring once again to Figure 22, method event table 1006 may in the preferred embodiment include from 1 to N method event records 1012. Each of these method event records 1012 corresponds to a different event the method 1000 represented by method core 1000' may respond to. Methods 1000 in the preferred embodiment may have completely different behavior depending upon the event they respond to. For example, an AUDIT method may store information in an audit trail UDE 1200 in response to an event corresponding to a user's use of an object or other resource. This same AUDIT method may report the stored audit trail to a VDE administrator or other participant in response to an administrative event such as, for example, a timer expiring within a VDE node or a request from another VDE participant to report the audit trail. In the preferred embodiment, each of these different events may be represented by an "event code." This "event code" may be passed as a parameter to a method when the method is called, and used to "look up" the appropriate method event record 1012 within method event table 1006. The selected method event record 1012, in turn, specifies the appropriate information (e.g., load module(s) 1100, data element UDE(s) and MDE(s) 1200, 1202, and/or PERC(s) 808) used to construct a component assembly 690 for execution in response to the event that has occurred.

Thus, in the preferred embodiment, each method event record 1012 may include an event field 1014, a LM/PERC reference field 1016, and any number of data reference fields 1018. Event fields 1014 in the preferred embodiment may contain a "event code" or other information identifying the corresponding event. The LM/PERC reference field 1016 may provide a reference into the secure database 610 (or other "pointer" information) identifying a load module 1100 and/or a PERC 808 providing (or referencing) executable code to be loaded and executed to perform the method in response to the event. Data reference fields 1018 may include information referencing a UDE 1200 or a MDE 1202. These data structures may be contained in the method local data area 1008 of the method core 1000', or they may be stored within the secure database 610 as independent deliverables.

The following table is an example of a possible more detailed field layout for a method event record 1012:

Field Type	Description
Event Field 1014	Identifies corresponding event.
Access tag	Secret tag to grant access to this row of the method event record.

20

5

10

Field Type		Description
LM/PERC Reference Field 1016	DB ID or offset/size	Database reference (or local pointer).
	Correlation tag	Correlation tag to assert when referencing this element.
# of Data Element Reference Fields		Count of data reference fields in the method event record.
Data Reference Field 1	UDE ID or offset/size	Database 610 reference (or local pointer).
	Correlation tag	Correlation tag to assert when referencing this element.
...		
Data Reference Field n	UDE ID or offset/size	Database 610 reference (or local pointer).
	Correlation tag	Correlation tag to assert when referencing this element.

15

Load Modules

Figure 23 is an example of a load module 1100 provided by the preferred embodiment. In general, load modules 1100 represent a collection of basic functions that are used for control operations.

20

Load module 1100 contains code and static data (that is functionally the equivalent of code), and is used to perform the basic operations of VDE 100. Load modules 1100 will generally

be shared by all the control structures for all objects in the system, though proprietary load modules are also permitted. Load modules 1100 may be passed between VDE participants in administrative object structures 870, and are usually stored in
5 secure database 610. They are always encrypted and authenticated in both of these cases. When a method core 1000' references a load module 1100, a load module is loaded into the SPE 503, decrypted, and then either passed to the electronic appliance microprocessor for executing in an HPE 655 (if that is
10 where it executes), or kept in the SPE (if that is where it executes). If no SPE 503 is present, the load module may be decrypted by the HPE 655 prior to its execution.

Load module creation by parties is preferably controlled by
15 a certification process or a ring based SPU architecture. Thus, the process of creating new load modules 1100 is itself a controlled process, as is the process of replacing, updating or deleting load modules already stored in a secured database 610.

20 A load module 1100 is able to perform its function only when executed in the protected environment of an SPE 503 or an HPE 655 because only then can it gain access to the protected elements (e.g., UDEs 1200, other load modules 1100) on which it operates. Initiation of load module execution in this

environment is strictly controlled by a combination of access tags, validation tags, encryption keys, digital signatures and/or correlation tags. Thus, a load module 1100 may only be referenced if the caller knows its ID and asserts the shared
5 secret correlation tag specific to that load module. The decrypting SPU may match the identification token and local access tag of a load module after decryption. These techniques make the physical replacement of any load module 1100 detectable at the next physical access of the load module.
10 Furthermore, load modules 1100 may be made "read only" in the preferred embodiment. The read-only nature of load modules 1100 prevents the write-back of load modules that have been tampered with in non-secure space.

15 Load modules are not necessarily directly governed by PERCs 808 that control them, nor must they contain any time/date information or expiration dates. The only control consideration in the preferred embodiment is that one or more methods 1000 reference them using a correlation tag (the value
20 of a protected object created by the load module's owner, distributed to authorized parties for inclusion in their methods, and to which access and use is controlled by one or more PERCs 808). If a method core 1000' references a load module 1100 and asserts the proper correlation tag (and the load module satisfies

the internal tamper checks for the SPE 503), then that load module can be loaded and executed, or it can be acquired from, shipped to, updated, or deleted by, other systems.

5 As shown in Figure 23, load modules 1100 in the preferred embodiment may be constructed of a public (unencrypted) header 802, a private (encrypted) header 804, a private body 1106 containing the encrypted executable code, and one or more data description elements ("DTDs") 1108. The DTDs 1108 may be
10 stored within a load module 1100, or they may be references to static data elements stored in secure database 610.

The following is an example of a possible field layout for load module public header 802:

15

Field Type		Description
LM ID		VDE ID of Load Module.
	Creator ID	Site ID of creator of this load module.
	Type ID	Constant indicates load module type.

Field Type		Description
	LM ID	Unique sequence number for this load module, which uniquely identifies the load module in a sequence of load modules created by an authorized VDE participant.
	Version ID	Version number of this load module.
Other classification information	Class ID	ID to support different load module classes.
	Type ID	ID to support method type compatible searching.
Descriptive Information	Description	Textual description of the load module.
	Execution space code	Value that describes what execution space (e.g., SPE or HPE) this load module.

5

10 Many load modules 1100 contain code that executes in an SPE 503. Some load modules 1100 contain code that executes in an HPE 655. This allows methods 1000 to execute in whichever environment is appropriate. For example, an INFORMATION method 1000 can be built to execute only in SPE 503 secure space for government classes of security, or in an HPE 655 for

15 commercial applications. As described above, the load module public header 802 may contain an "execution space code" field

that indicates where the load module 1100 needs to execute.

This functionality also allows for different SPE instruction sets as well as different user platforms, and allows methods to be constructed without dependencies on the underlying load module instruction set.

Load modules 1100 operate on three major data areas: the stack, load module parameters, and data structures. The stack and execution memory size required to execute the load module 1100 are preferably described in private header 804, as are the data descriptions from the stack image on load module call, return, and any return data areas. The stack and dynamic areas are described using the same DTD mechanism. The following is an example of a possible layout for a load module private header 1104:

Field Type		Description
Copy of some or all of information from public header 802		Object ID from Public Header.
Other classification information	Check Value	Check Value for Public Header.
Descriptive Information	LM Size	Size of executable code block.
	LM Exec Size	Executable code size for the load module.
	LM Exec Stack	Stack size required for the load module.

	Execution space code	Code that describes the execution space for this load module.
Access and reference tags	Access tag	Tags used to determine if the load module is the correct LM requested by the SPE.
	Validation tag	
	Correlation tag	Tag used to determine if the caller of the LM has the right to execute this LM.
	Digital Signature	Used to determine if the LM executable content is intact and was created by a trusted source (one with a correct certificate for creating LMs).
Data record descriptor information	DTD count	Number of DTDs that follow the code block.
	DTD 1 reference	<p>If locally defined, the physical size and offset in bytes of the first DTD defined for this LM.</p> <p>If publicly referenced DTD, this is the DTD ID and the correlation tag to permit access to the record.</p>
	...	
	DTD N reference	<p>If locally defined, the physical size and offset in bytes of the Nth DTD defined for this LM.</p> <p>If publicly referenced DTD, this is the DTD ID and the correlation tag to permit access to the record.</p>
Check Value		Check Value for entire LM.

Each load module 1100 also may use DTD 1108

information to provide the information necessary to support building methods from a load module. This DTD information

contains the definition expressed in a language such as SGML for the names and data types of all of the method data fields that the load module supports, and the acceptable ranges of values that can be placed in the fields. Other DTDs may describe the function of the load module 1100 in English for inclusion in an electronic contract, for example.

The next section of load module 1100 is an encrypted executable body 1106 that contains one or more blocks of encrypted code. Load modules 1100 are preferably coded in the "native" instruction set of their execution environment for efficiency and compactness. SPU 500 and platform providers may provide versions of the standard load modules 1100 in order to make their products cooperate with the content in distribution mechanisms contemplated by VDE 100. The preferred embodiment creates and uses native mode load modules 1100 in lieu of an interpreted or "p-code" solution to optimize the performance of a limited resource SPU. However, when sufficient SPE (or HPE) resources exist and/or platforms have sufficient resources, these other implementation approaches may improve the cross platform utility of load module code.

The following is an example of a field layout for a load module DTD 1108:

5

Field Type		Description
DTD ID		Uses Object ID from Private Header.
	Creator ID	Site ID of creator of this DTD.
	Type ID	Constant.
	DTD ID	Unique sequence number for this DTD.
	Version ID	Version number of this DTD.
Descriptive Information	DTD Size	Size of DTD block.
Access and reference tags	Access tag	Tags used to determine if the DTD is the correct DTD requested by the SPE.
	Validation tag	
	Correlation tag	Tag used to determine if the caller of this DTD has the right to use the DTD.
DTD Body	DTD Data Definition 1	
	DTD Data Definition 2	
	⋮	
	DTD Data Definition N	
	Check Value	Check Value for entire DTD record.

10

Some examples of how load modules 1100 may use DTDs 1108 include:

15

- Increment data element (defined by name in DTD3) value in data area DTD4 by value in DTD1

- Set data element (defined by name in DTD3) value in data area DTD4 to value in DTD3
- 5 • Compute atomic element from event in DTD1 from table in DTD3 and return in DTD2
- Compute atomic element from event in DTD1 from equation in DTD3 and return in DTD2
- 10 • Create load module from load module creation template referenced in DTD3
- Modify load module in DTD3 using content in DTD4
- 15 • Destroy load module named in DTD3

Commonly used load modules 1100 may be built into a SPU 500 as space permits. VDE processes that use built-in load modules 1100 will have significantly better performance than
20 processes that have to find, load and decrypt external load modules. The most useful load modules 1100 to build into a SPU might include scaler meters, fixed price billing, budgets and load modules for aggregate methods that perform these three
processes.

25

User Data Elements (UDEs) 1200 and Method Data Elements (MDEs) 1202

User Data Elements (UDEs) 1200 and Method Data
30 Elements (MDEs) 1202 in the preferred embodiment store data.

There are many types of UDEs 1200 and MDEs 1202 provided by the preferred embodiment. In the preferred embodiment, each of these different types of data structures shares a common overall format including a common header definition and naming

5 scheme. Other UDEs 1200 that share this common structure include "local name services records" (to be explained shortly) and account information for connecting to other VDE participants. These elements are not necessarily associated with an individual user, and may therefore be considered MDEs 1202.

10 All UDEs 1200 and all MDEs 1202 provided by the preferred embodiment may, if desired, (as shown in Figure 16) be stored in a common physical table within secure database 610, and database access processes may commonly be used to access all of these different types of data structures.

15

In the preferred embodiment, PERCs 808 and user rights table records are types of UDE 1200. There are many other types of UDEs 1200/MDEs 1202, including for example, meters, meter trails, budgets, budget trails, and audit trails. Different

20 formats for these different types of UDEs/MDEs are defined, as described above, by SGML definitions contained within DTDs 1108. Methods 1000 use these DTDs to appropriately access UDEs/MDEs 1200, 1202.

Secure database 610 stores two types of items: static and dynamic. Static data structures and other items are used for information that is essentially static information. This includes load modules 1100, PERCs 808, and many components of methods. These items are not updated frequently and contain expiration dates that can be used to prevent "old" copies of the information from being substituted for newly received items. These items may be encrypted with a site specific secure database file key when they are stored in the secure database 610, and then decrypted using that key when they are loaded into the SPE.

Dynamic items are used to support secure items that must be updated frequently. The UDEs 1200 of many methods must be updated and written out of the SPE 503 after each use. Meters and budgets are common examples of this. Expiration dates cannot be used effectively to prevent substitution of the previous copy of a budget UDE 1200. To secure these frequently updated items, a transaction tag is generated and included in the encrypted item each time that item is updated. A list of all VDE item IDs and the current transaction tag for each item is maintained as part of the secure database 610.

Figure 24 shows an example of a user data element

("UDE") 1200 provided by the preferred embodiment. As shown in Figure 24, UDE 1200 in the preferred embodiment includes a public header 802, a private header 804, and a data area 1206.

5 The layout for each of these user data elements 1200 is generally defined by an SGML data definition contained within a DTD 1108 associated with one or more load modules 1100 that operate on the UDE 1200.

10 UDEs 1200 are preferably encrypted using a site specific key once they are loaded into a site. This site-specific key masks a validation tag that may be derived from a cryptographically strong pseudo-random sequence by the SPE 503 and updated each time the record is written back to the secure database 610.

15 This technique provides reasonable assurance that the UDE 1200 has not been tampered with nor substituted when it is requested by the system for the next use.

Meters and budgets are perhaps among the most common

20 data structures in VDE 100. They are used to count and record events, and also to limit events. The data structures for each meter and budget are determined by the content provider or a distributor/redistributor authorized to change the information. Meters and budgets, however, generally have common

information stored in a common header format (e.g., user ID, site ID and related identification information).

The content provider or distributor/redistributor may specify data structures for each meter and budget UDE.

Although these data structures vary depending upon the particular application, some are more common than others. The following table lists some of the more commonly occurring data structures for METER and BUDGET methods:

Field type	Format	Typical Use	Description or Use
Ascending Use Counter	byte, short, long, or unsigned versions of the same widths	Meter/Budget	Ascending count of uses.
Descending Use Counter	byte, short, long, or unsigned versions of the same widths	Budget	Descending count of permitted use; eg., remaining budget.
Counter/Limit	2, 4 or 8 byte integer split into two related bytes or words	Meter/Budget	usage limits since a specific time; generally used in compound meter data structures.
Bitmap	Array bytes	Meter/Budget	Bit indicator of use or ownership.
Wide bitmap	Array of bytes	Meter/Budget	Indicator of use or ownership that may age with time.
Last Use Date	time t	Meter/Budget	Date of last use.

Field type	Format	Typical Use	Description or Use
Start Date	time_t	Budget	Date of first allowable use.
Expiration Date	time_t	Meter/Budget	Expiration Date.
Last Audit Date	time_t	Meter/Budget	Date of last audit.
Next Audit Date	time_t	Meter/Budget	Date of next required audit.
Auditor	VDE ID	Meter/Budget	VDE ID of authorized auditor.

5
 10 The information in the table above is not complete or comprehensive, but rather is intended to show some examples of types of information that may be stored in meter and budget related data structures. The actual structure of particular meters and budgets is determined by one or more DTDs 1108 associated with the load modules 1100 that create and
 15 manipulate the data structure. A list of data types permitted by the DTD interpreter 590 in VDE 100 is extensible by properly authorized parties.

20 Figure 25 shows an example of one particularly advantageous kind of UDE 1200 data area 1206. This data area 1206 defines a "map" that may be used to record usage information. For example, a meter method 1000 may maintain one or more "usage map" data areas 1206. The usage map may

be a "usage bit map" in the sense that it stores one or more bits of information (i.e., a single or multi-dimensional bit image) corresponding to each of several types or categories of usage.

Usage maps are an efficient means for referencing prior usage.

5 For example, a usage map data area may be used by a meter method 1000 to record all applicable portions of information content that the user has paid to use, thus supporting a very efficient and flexible means for allowing subsequent user usage of the same portions of the information content. This may enable
10 certain VDE related security functions such as "contiguousness," "logical relatedness," randomization of usage, and other usage types. Usage maps may be analyzed for other usage patterns (e.g., quantity discounting, or for enabling a user to reaccess information content for which the user previously paid for
15 unlimited usage).

The "usage map" concept provided by the preferred embodiment may be tied to the concept of "atomic elements." In the preferred embodiment, usage of an object 300 may be
20 metered in terms of "atomic elements." In the preferred embodiment, an "atomic element" in the metering context defines a unit of usage that is "sufficiently significant" to be recorded in a meter. The definition of what constitutes an "atomic element" is determined by the creator of an object 300. For instance, a

"byte" of information content contained in an object 300 could be defined as an "atomic element," or a record of a database could be defined as an "atomic element," or each chapter of an electronically published book could be defined as an "atomic element."

An object 300 can have multiple sets of overlapping atomic elements. For example, an access to any database in a plurality of databases may be defined as an "atomic element."

Simultaneously, an access to any record, field of records, sectors of informations, and/or bytes contained in any of the plurality of databases might also be defined as an "atomic element." In an electronically published newspaper, each hundred words of an article could be defined as an "atomic element," while articles of more than a certain length could be defined as another set of "atomic elements." Some portions of a newspaper (e.g., advertisements, the classified section, etc.) might not be mapped into an atomic element.

The preferred embodiment provides an essentially unbounded ability for the object creator to define atomic element types. Such atomic element definitions may be very flexible to accommodate a wide variety of different content usage. Some examples of atomic element types supported by the preferred

embodiment include bytes, records, files, sectors, objects, a quantity of bytes, contiguous or relatively contiguous bytes (or other predefined unit types), logically related bytes containing content that has some logical relationship by topic, location or other user specifiable logic of relationship, etc. Content creators preferably may flexibly define other types of atomic elements.

The preferred embodiment of the present invention provides EVENT methods to provide a mapping between usage events and atomic elements. Generally, there may be an EVENT method for each different set of atomic elements defined for an object 300. In many cases, an object 300 will have at least one type of atomic element for metering relating to billing, and at least one other atomic element type for non-billing related metering (e.g., used to, for example, detect fraud, bill advertisers, and/or collect data on end user usage activities).

In the preferred embodiment, each EVENT method in a usage related context performs two functions: (1) it maps an accessed event into a set of zero or more atomic elements, and (2) it provides information to one or more METER methods for metering object usage. The definition used to define this mapping between access events and atomic elements may be in the form of a mathematical definition, a table, a load module, etc.

When an EVENT method maps an access request into "zero" atomic elements, a user accessed event is not mapped into any atomic element based on the particular atomic element definition that applies. This can be, for example, the object owner is not
5 interested in metering usage based on such accesses (e.g., because the object owner deems such accesses to be insignificant from a metering standpoint).

A "usage map" may employ a "bit map image" for storage
10 of usage history information in a highly efficient manner. Individual storage elements in a usage map may correspond to atomic elements. Different elements within a usage map may correspond to different atomic elements (e.g., one map element may correspond to number of bytes read, another map element
15 may correspond to whether or not a particular chapter was opened, and yet another map element may correspond to some other usage event).

One of the characteristics of a usage map provided by the
20 preferred embodiment of the present invention is that the significance of a map element is specified, at least in part, by the position of the element within the usage map. Thus, in a usage map provided by the preferred embodiment, the information indicated or encoded by a map element is a function of its

position (either physically or logically) within the map structure.

As one simple example, a usage map for a twelve-chapter novel could consist of twelve elements, one for each chapter of the novel. When the user opens the first chapter, one or more bits within the element corresponding to the first chapter could be changed in value (e.g., set to "one"). In this simple example where the owner of the content object containing the novel was interested only in metering which chapters had been opened by the user, the usage map element corresponding to a chapter could be set to "one" the first time the user opened that corresponding chapter, and could remain "one" no matter how many additional times the user opened the chapter. The object owner or other interested VDE participant would be able to rapidly and efficiently tell which chapter(s) had been opened by the user simply by examining the compact usage map to determine which elements were set to "one."

Suppose that the content object owner wanted to know how many times the user had opened each chapter of the novel. In this case, the usage map might comprise, for a twelve-chapter novel, twelve elements each of which has a one-to-one correspondence with a different one of the twelve chapters of the novel. Each time a user opens a particular chapter, the corresponding METER method might increment the value

contained in the corresponding usage map element. In this way, an account could be readily maintained for each of the chapters of the novel.

5 The position of elements within a usage map may encode a multi-variable function. For example, the elements within a usage map may be arranged in a two-dimensional array as shown in Figure 25B. Different array coordinates could correspond to independent variables such as, for example, atomic
10 elements and time. Suppose, as an example, that a content object owner distributes an object containing a collection of audio recordings. Assume further that the content object owner wants to track the number of times the user listens to each recording within the collection, and also wants to track usage based on
15 month of the year. Thus, assume that the content object owner wishes to know how many times the user during the month of January listened to each of the recordings on a recording-by-recording basis, similarly wants to know this same information for the month of February, March, etc. In this case, the usage
20 map (see Figure 25B) might be defined as a two-dimensional array of elements. One dimension of the array might encode audio recording number. The other dimension of the array might encode month of the year. During the month of January, the corresponding METER method would increment elements in the

array in the "January" column of the array, selecting which
element to increment as a function of recording number. When
January comes to an end, the METER method might cease
writing into the array elements in the January column, and
5 instead write values into a further set of February array
elements—once again selecting the particular array element in
this column as a function of recording number. This concept may
be extended to N dimensions encoding N different variables.

10 Usage map meters are thus an efficient means for
referencing prior usage. They may be used to enable certain
VDE related security functions such as testing for
contiguousness (including relative contiguousness), logical
relatedness (including relative logical relatedness), usage
15 randomization, and other usage patterns. For example, the
degree or character of the "randomness" of content usage by a
user might serve as a potential indicator of attempts to
circumvent VDE content budget limitations. A user or groups of
users might employ multiple sessions to extract content in a
20 manner which does not violate contiguousness, logical
relatedness or quantity limitations, but which nevertheless
enables reconstruction of a material portion or all of a given,
valuable unit of content. Usage maps can be analyzed to
determine other patterns of usage for pricing such as, for

example, quantity discounting after usage of a certain quantity of any or certain atomic units, or for enabling a user to reaccess an object for which the user previously paid for unlimited accesses (or unlimited accesses over a certain time duration).

5 Other useful analyses might include discounting for a given atomic unit for a plurality of uses.

10 A further example of a map meter includes storing a record of all applicable atomic elements that the user has paid to use (or alternatively, has been metered as having used, though payment may not yet have been required or made). Such a usage map would support a very efficient and flexible way to allow subsequent user usage of the same atomic elements.

15 A further usage map could be maintained to detect fraudulent usage of the same object. For example, the object might be stored in such a way that sequential access of long blocks should never occur. A METER method could then record all applicable atomic elements accesses during, for example, any
20 specified increment of time, such as ten minutes, an hour, a day, a month, a year, or other time duration). The usage map could be analyzed at the end of the specified time increment to check for an excessively long contiguous set of accessed blocks, and/or could be analyzed at the initiation of each access to applicable

atomic elements. After each time duration based analysis, if no fraudulent use is detected, the usage map could be cleared (or partially cleared) and the mapping process could begin in whole or in part anew. If a fraudulent use pattern is suspected or
5 detected, that information might be recorded and the use of the object could be halted. For example, the user might be required to contact a content provider who might then further analyze the usage information to determine whether or not further access should be permitted.

10

Figure 25c shows a particular type of "wide bit map" usage record 1206 wherein each entry in the usage record corresponds to usage during a particular time period (e.g., current month usage, last month's usage, usage in the month before last, etc.).

15

The usage record shown thus comprises an array of "flags" or fields 1206, each element in the array being used to indicate usage in a different time period in this particular example.

20

When a time period ends, all elements 1206 in the array may be shifted one position, and thus usage information (or the purchase of user access rights) over a series of time periods can be reflected by a series of successive array elements. In the specific example shown in Figure 25c, the entire wide array 1206 is shifted by one array position each month, with the oldest array element being deleted and the new array element being "turned"

in a new array map corresponding to the current time period. In this example, record 1302 tracks usage access rights and/or other usage related activities during the present calendar month as well for the five immediately prior calendar months.

5 Corresponding billing and/or billing method 406 may inspect the map, determine usage as related to billing and/or security monitoring for current usage based on a formula that employs the usage data stored in the record, and updates the wide record to indicate the applicable array elements for which usage
10 occurred or the like. A wide bit map may also be used for many other purposes such as maintaining an element by element count of usage, or the contiguousness, relatedness, etc. function described above, or some combination of functionality.

15 Audit trail maps may be generated at any frequency determined by control, meter, budget and billing methods and load modules associated with those methods. Audit trails have a similar structure to meters and budgets and they may contain user specific information in addition to information about the
20 usage event that caused them to be created. Like meters and budgets, audit trails have a dynamic format that is defined by the content provider or their authorized designee, and share the basic element types for meters and budgets shown in the table above. In addition to these types, the following table lists some

examples of other significant data fields that may be found in
audit trails:

	Field type	Format	Typical Use	Description of Use
5	Use Event ID	unsigned long	Meter/Budget/ Billing	Event ID that started a processing sequence.
	Internal Sequence Number	unsigned long	Meter/Budget/ Billing	Transaction number to help detect audits that have been tampered with.
10	Atomic Element(s) & Object ID	Unsigned integer(s) of appropriate width	Meter/Billing	Atomic element(s) and ID of object that was used.
	Personal User Information	Character or other information	Budget/Billing	Personal information about user.
	Use Date/Time	time_t	Meter/Budget/ Billing	Date/time of use.
15	Site ID/User ID	VDE ID	Meter/Budget/ Billing	VDE ID of user.

Audit trail records may be automatically combined into
single records to conserve header space. The combination
process may, for example, occur under control of a load module
that creates individual audit trail records.

Permissions Record Overview

Figure 16 also shows that PERCs 808 may be stored as
part of secure database 610. Permissions records ("PERCs") 808

are at the highest level of the data driven control hierarchy provided by the preferred embodiment of VDE 100. Basically, there is at least one PERC 808 that corresponds to each information and/or transactional content distributed by VDE 100. Thus, at least one PERC 808 exists for each VDE object 300 in the preferred embodiment. Some objects may have multiple corresponding PERCs 808. PERC 808 controls how access and/or manipulation permissions are distributed and/or how content and/or other information may otherwise be used. PERC 808 also specifies the "rights" of each VDE participant in and to the content and/or other information.

In the preferred embodiment, no end user may use or access a VDE object unless a permissions record 808 has been delivered to the end user. As discussed above, a PERC 808 may be delivered as part of a traveling object 860 or it may be delivered separately (for example, within an administrative object). An electronic appliance 600 may not access an object unless a corresponding PERC 808 is present, and may only use the object and related information as permitted by the control structures contained within the PERC.

Briefly, the PERC 808 stores information concerning the methods, method options, decryption keys and rights with respect to a corresponding VDE object 300.

5 PERC 808 includes control structures that define high level categories or classifications of operations. These high level categories are referred to as "rights." The "right" control structures, in turn, provide internal control structures that reference "methods" 1000. The internal structure of preferred
10 embodiment PERC 808 organizes the "methods" that are required to perform each allowable operation on an object or associated control structure (including operations performed on the PERC itself). For example, PERC 808 contains decryption
15 keys for the object, and usage of the keys is controlled by the methods that are required by the PERC for performing operations associated with the exercise of a "right."

 PERC 808 for an object is typically created when the object is created, and future substantive modifications of a PERC, if
20 allowed, are controlled by methods associated with operations using the distribution right(s) defined by the same (or different) PERC.

Figure 22 shows the internal structures present in an example of a PERC 808 provided by the preferred embodiment. All of the structures shown represent (or reference) collections of methods required to process a corresponding object in some specific way. PERCs 808 are organized as a hierarchical structure, and the basic elements of the hierarchy are as follows:

"rights" records 906

"control sets" 914

"required method" records 920 and

"required method options" 924.

There are other elements that may be included in a PERC 808 hierarchy that describe rules and the rule options to support the negotiation of rule sets and control information for smart objects and for the protection of a user's personal information by a privacy filter. These alternate elements may include:

optional rights records

optional control sets

optional method records

permitted rights records

permitted rights control sets

permitted method records

required DTD descriptions

optional DTD descriptions

permitted DTD descriptions

These alternate fields can control other processes that may, in part, base negotiations or decisions regarding their operation on the contents of these fields. Rights negotiation, smart object
5 control information, and related processes can use these fields for more precise control of their operation.

The PERC 808 shown in Figure 26 includes a PERC header 900, a CS0 ("control set 0") 902, private body keys 904,
10 and one or more rights sub-records 906. Control set 0 902 in the preferred embodiment contains information that is common to one or more "rights" associated with an object 300. For example, a particular "event" method or methods might be the same for usage rights, extraction rights and/or other rights. In that case,
15 "control set 0" 902 may reference this event that is common across multiple "rights." The provision of "control set 0" 902 is actually an optimization, since it would be possible to store different instances of a commonly-used event within each of plural "rights" records 906 of a PERC 808.

20

Each rights record 906 defines a different "right" corresponding to an object. A "right" record 906 is the highest level of organization present in PERC 808. There can be several different rights in a PERC 808. A "right" represents a major

functional partitioning desired by a participant of the basic architecture of VDE 100. For example, the right to use an object and the right to distribute rights to use an object are major functional groupings within VDE 100. Some examples of possible rights include access to content, permission to distribute rights to access content, the ability to read and process audit trails related to content and/or control structures, the right to perform transactions that may or may not be related to content and/or related control structures (such as banking transactions, catalog purchases, the collection of taxes, EDI transactions, and such), and the ability to change some or all of the internal structure of PERCs created for distribution to other users. PERC 808 contains a rights record 906 for each type of right to object access/use the PERC grants.

15

Normally, for VDE end users, the most frequently granted right is a usage right. Other types of rights include the "extraction right," the "audit right" for accessing audit trail information of end users, and a "distribution right" to distribute an object. Each of these different types of rights may be embodied in a different rights record 906 (or alternatively, different PERCs 808 corresponding to an object may be used to grant different rights).

20

Each rights record 906 includes a rights record header 908,
a CSR ("control set for right") 910, one or more "right keys" 912,
and one or more "control sets" 914. Each "rights" record 906
contains one or more control sets 914 that are either required or
selectable options to control an object in the exercise of that
"right." Thus, at the next level, inside of a "right" 906, are control
sets 914. Control sets 914, in turn, each includes a control set
header 916, a control method 918, and one or more required
methods records 920. Required methods records 920, in turn,
each includes a required method header 922 and one or more
required method options 924.

Control sets 914 exist in two types in VDE 100: common
required control sets which are given designations "control set 0"
or "control set for right," and a set of control set options. "Control
set 0" 902 contains a list of required methods that are common to
all control set options, so that the common required methods do
not have to be duplicated in each control set option. A "control
set for right" ("CSR") 910 contains a similar list for control sets
within a given right. "Control set 0" and any "control sets for
rights" are thus, as mentioned above, optimizations; the same
functionality for the control sets can be accomplished by listing
all the common required methods in each control set option and
omitting "control set 0" and any "control sets for rights."

One of the control set options, "control set 0" and the appropriate "control set for right" together form a complete control set necessary to exercise a right.

5 Each control set option contains a list of required methods 1000 and represents a different way the right may be exercised. Only one of the possible complete control sets 914 is used at any one time to exercise a right in the preferred embodiment.

10 Each control set 914 contains as many required methods records 920 as necessary to satisfy all of the requirements of the creators and/or distributors for the exercise of a right. Multiple ways a right may be exercised, or multiple control sets that govern how a given right is exercised, are both supported. As an
15 example, a single control set 914 might require multiple meter and budget methods for reading the object's content, and also require different meter and budget methods for printing an object's content. Both reading and printing an object's content can be controlled in a single control set 914.

20

Alternatively, two different control set options could support reading an object's content by using one control set option to support metering and budgeting the number of bytes read, and the other control set option to support metering and

budgeting the number of paragraphs read. One or the other of these options would be active at a time.

Typically, each control set 914 will reference a set of related methods, and thus different control sets can offer a different set of method options. For example, one control set 914 may represent one distinct kind of metering methodology, and another control set may represent another, entirely different distinct metering methodology.

At the next level inside a control set 914 are the required methods records 920. Methods records 920 contain or reference methods 1000 in the preferred embodiment. Methods 1000 are a collection of "events," references to load modules associated with these events, static data, and references to a secure database 610 for automatic retrieval of any other separately deliverable data elements that may be required for processing events (e.g., UDEs). A control set 914 contains a list of required methods that must be used to exercise a specific right (i.e., process events associated with a right). A required method record 920 listed in a control set 914 indicates that a method must exist to exercise the right that the control set supports. The required methods may reference "load modules" 1100 to be discussed below.

Briefly, load modules 1100 are pieces of executable code that may be used to carry out required methods.

Each control set 914 may have a control method record 918
5 as one of its required methods. The referenced control method
may define the relationships between some or all of the various
methods 1000 defined by a control set 906. For example, a
control method may indicate which required methods are
functionally grouped together to process particular events, and
10 the order for processing the required methods. Thus, a control
method may specify that required method referenced by record
920(a)(1)(i) is the first to be called and then its output is to go to
required method referenced by record 920(a)(1)(ii) and so on. In
this way, a meter method may be tied to one or more billing
15 methods and then the billing methods may be individually tied
to different budget methods, etc.

Required method records 920 specify one or more required
method options 924. Required method options are the lowest
20 level of control structure in a preferred embodiment PERC 808.
By parameterizing the required methods and specifying the
required method options 924 independently of the required
methods, it becomes possible to reuse required methods in many
different circumstances.

For example, a required method record 920 may indicate that an actual budget method ID must be chosen from the list of budget method IDs in the required method option list for that required method. Required method record 920 in this case does not contain any method IDs for information about the type of method required, it only indicates that a method is required. Required method option 924 contains the method ID of the method to be used if this required method option is selected. As a further optimization, an actual method ID may be stored if only one option exists for a specific required method. This allows the size of this data structure to be decreased.

PERC 808 also contains the fundamental decryption keys for an object 300, and any other keys used with "rights" (for encoding and/or decoding audit trails, for example). It may contain the keys for the object content or keys to decrypt portions of the object that contain other keys that then can be used to decrypt the content of the object. Usage of the keys is controlled by the control sets 914 in the same "right" 906 within PERC 808.

In more detail, Figure 26 shows PERC 808 as including private body keys 904, and right keys 912. Private body keys 904 are used to decrypt information contained within a private

body 806 of a corresponding VDE object 300. Such information may include, for example, methods 1000, load modules 1100 and/or UDEs 1200, for example. Right keys 912 are keys used to exercise a right in the preferred embodiment. Such right keys 912 may include, for example, decryption keys that enable a method specified by PERC 808 to decrypt content for release by a VDE node to an end user. These right keys 912 are, in the preferred embodiment, unique to an object 300. Their usage is preferably controlled by budgets in the preferred embodiment.

Detailed Example of a PERC 808

Figures 26A and 26B show one example of a preferred embodiment PERC 808. In this example, PERC header 900 includes:

- a site record number 926,
- a field 928 specifying the length of the private body key block,
- a field 930 specifying the length of the PERC,
- an expiration date/time field 932 specifying the expiration date and/or time for the PERC,
- a last modification date/time field 934 specifying the last date and/or time the PERC 808 was modified,

the original distributor ID field 936 that specifies
who originally distributed the PERC and/or
corresponding object,
a last distributor field 938 that specifies who was
5 the last distributor of the PERC and/or the
object,
an object ID field 940 identifying the corresponding
VDE object 300,
a field 942 that specifies the class and/or type of
10 PERC and/or the instance ID for the record
class to differentiate the PERCs of the same
type that may differ in their particulars,
a field 944 specifying the number of "rights" sub-
records 906 within the PERC, and
15 a validation tag 948.

The PERC 808 shown in Figures 26a, 26b also has private body
keys stored in a private body key block 950.

This PERC 808 includes a control set 0 sub-record 914 (0)
20 that may be used commonly by all of rights 906 within the
PERC. This control set 0 record 914(0) may include the following
fields:

a length field 952 specifying the length of the control
set 0 record

a field 954 specifying the number of required
method records 920 within the control set
an access tag field 956 specifying an access tag to
control modification of the record and
5 one or more required method records 920.

Each required method record 920, in turn may include:

a length field 958 specifying the length of the
required method record
a field 960 specifying the number of method option
10 records within the required method record 920
an access tag field 962 specifying an access tag to
control modification of the record and
one or more method option records 924.

Each method option sub-record 924 may include:

15 a length field 964 specifying the length of the
method option record
a length field 966 specifying the length of the data
area (if any) corresponding to the method
option record
20 a method ID field 968 specifying a method ID (e.g.,
type/owner/class/instance)
a correlation tag field 970 specifying a correlation
tag for correlating with the method specified
in field 968

an access tag field 972 specifying an access tag to
control modification of this record
a method-specific attributes field 974
a data area 976 and
5 a check value field 978 for validation purposes

In this example of PERC 808 also includes one or more
rights records 906, and an overall check value field 980. Figure
23b is an example of one of right records 906 shown in Figure
10 16a. In this particular example, rights record 906a includes a
rights record header 908 comprising:

a length field 982 specifying the length of the rights
key block 912
a length field 984 specifying the length of the rights
15 record 908
an expiration date/time field 986 specifying the
expiration date and/or time for the rights
record
a right ID field 988 identifying a right
20 a number field 990 specifying the number of control
sets 914 within the rights record 906, and
an access tag field 992 specifying an access tag to
control modification of the right record.

This example of rights record 906 includes:

a control set for this right (CSR) 910

a rights key block 912

one or more control sets 914, and

5 a check value field 994.

Object Registry

Referring once again to Figure 16, secure database 610 provides data structures that support a "lookup" mechanism for
10 "registered" objects. This "lookup" mechanism permits electronic appliance 600 to associate, in a secure way, VDE objects 300 with PERCs 808, methods 1000 and load modules 1100. In the preferred embodiment, this lookup mechanism is based in part on data structures contained within object registry 450.

15

In one embodiment, object registry 450 includes the following tables:

- an object registration table 460;
- a subject table 462;
- 20 • a User Rights Table ("URT") 464;
- an Administrative Event Log 442;
- a shipping table 444; and
- a receiving table 446.

Object registry 460 in the example embodiment is a database of information concerning registered VDE objects 300 and the rights of users and user groups with regard to those objects. When electronic appliance 600 receives an object 300
5 containing a new budget or load module 1100, the electronic appliance usually needs to add the information contained by the object to secure database 610. Moreover, when any new VDE object 300 arrives at an electronic appliance 600, the electronic appliance must "register" the object within object registry 450 so
10 that it can be accessed. The lists and records for a new object 300 are built in the preferred embodiment when the object is "registered" by the electronic appliance 600. The information for the object may be obtained from the object's encrypted private header, object body, and encrypted name services record. This
15 information may be extracted or derived from the object 300 by SPE 503, and then stored within secure database 610 as encrypted records.

In one embodiment, object registration table 460 includes
20 information identifying objects within object storage (repository) 728. These VDE objects 300 stored within object storage 728 are not, in the example embodiment, necessarily part of secure database 610 since the objects typically incorporate their own security (as necessary and required) and are maintained using

different mechanisms than the ones used to maintain the secure database. Even though VDE objects 300 may not strictly be part of secure database 610, object registry 450 (and in particular, object registration table 460) refers to the objects and thus

5 "incorporates them by reference" into the secure database. In the preferred embodiment, an electronic appliance 600 may be disabled from using any VDE object 300 that has not been appropriately registered with a corresponding registration record stored within object registration table 460.

10

Subject table 462 in the example embodiment establishes correspondence between objects referred to by object registration table 460 and users (or groups of users) of electronic appliance 600. Subject table 462 provides many of the attributes of an

15 access control list ("ACL"), as will be explained below.

User rights table 464 in the example embodiment provides permissioning and other information specific to particular users or groups of users and object combinations set forth in subject

20 table 462. In the example embodiment, permissions records 808 (also shown in Figure 16 and being stored within secure database 610) may provide a universe of permissioning for a particular object-user combination. Records within user rights table 464 may specify a sub-set of this permissioning universe

based on, for example, choices made by users during interaction at time of object registration.

Administrative event log 442, shipping table 444, and receiving table 446 provide information about receipts and deliveries of VDE objects 300. These data structures keep track of administrative objects sent or received by electronic appliance 600 including, for example, the purpose and actions of the administrative objects in summary and detailed form. Briefly, shipping table 444 includes a shipping record for each administrative object sent (or scheduled to be sent) by electronic appliance 600 to another VDE participant. Receiving table 446 in the preferred embodiment includes a receiving record for each administrative object received (or scheduled to be received) by electronic appliance 600. Administrative event log 442 includes an event log record for each shipped and each received administrative object, and may include details concerning each distinct event specified by received administrative objects.

Administrative Object Shipping and Receiving

Figure 27 is an example of a detailed format for a shipping table 444. In the preferred embodiment, shipping table 444 includes a header 444A and any number of shipping records 445. Header 444A includes information used to maintain shipping

table 444. Each shipping record 445 within shipping table 444 provides details concerning a shipping event (i.e., either a completed shipment of an administrative object to another VDE participant, or a scheduled shipment of an administrative object).

In the example embodiment of the secure database 610, shipping table header 444A may include a site record number 444A(1), a user (or group) ID 444A(2), a series of reference fields 444A(3)-444A(6), validation tags 444A(7)-444A(8), and a check value field 444A(9). The fields 444A(3)-444A(6) reference certain recent IDs that designate lists of shipping records 445 within shipping table 444. For example, field 444A(3) may reference to a "first" shipping record representing a completed outgoing shipment of an administrative object, and field 444A(4) may reference to a "last" shipping record representing a completed outgoing shipment of an administrative object. In this example, "first" and "last" may, if desired, refer to time or order of shipment as one example. Similarly, fields 444A(5) and 444A(6) may reference to "first" and "last" shipping records for scheduled outgoing shipments. Validation tag 444A(7) may provide validation from a name services record within name services record table 452 associated with the user (group) ID in the header. This permits access from the shipping record back to the

name services record that describes the sender of the object described by the shipping records. Validation tag 444A(8) provides validation for a "first" outgoing shipping record referenced by one or more of pointers 444A(3)-444A(6). Other
5 validation tags may be provided for validation of scheduled shipping record(s).

Shipping record 444(1) shown includes a site record number 445(1)(A). It also includes first and last scheduled
10 shipment date/times 445(1)(B), 445(1)(C) providing a window of time used for scheduling administrative object shipments. Field 445(1)(D) may specify an actual date/time of a completed shipment of an administrative object. Field 445(1)(E) provides
15 an ID of an administrative object shipped or to be shipped, and thus identifies which administrative object within object storage 728 pertains to this particular shipping record. A reference field 445(1)(G) references a name services record within name services record table 452 specifying the actual or intended recipient of the administrative object shipped or to be shipped. This information
20 within name services record table 452 may, for example, provide routing information sufficient to permit outgoing administrative objects manager 754 shown in Figure 12 to inform object switch 734 to ship the administrative object to the intended recipient. A field 445(1)(H) may specify (e.g., using a series of bit flags) the

purpose of the administrative object shipment, and a field 445(1)(I) may specify the status of the shipment. Reference fields 445(1)(J), 445(1)(K) may reference "previous" and "next" shipping records 445 in a linked list (in the preferred embodiment, there may be two linked lists, one for completed shipping records and the other for scheduled shipping records). Fields 445(1)(L) - 445(1)(P) may provide validation tags respectively from header 444A, to a record within administrative event log 442 pointed to by pointer 445(1)(F); to the name services record referenced by field 445(1)(G); from the previous record referenced by 445(1)(J); and to the next record referenced by field 445(1)(K). A check value field 445(1)(Q) may be used for validating shipping record 445.

Figure 28 shows an example of one possible detailed format for a receiving table 446. In one embodiment, receiving table 446 has a structure that is similar to the structure of the shipping table 444 shown in Figure 27. Thus, for example, receiving table 446 may include a header 446a and a plurality of receiving records 447, each receiving record including details about a particular reception or scheduled reception of an administrative object. Receiving table 446 may include two linked lists, one for completed receptions and another for schedule receptions. Receiving table records 447 may each

reference an entry within name services record table 452 specifying an administrative object sender, and may each point to an entry within administrative event log 442. Receiving records 447 may also include additional details about scheduled and/or completed reception (e.g., scheduled or actual date/time of reception, purpose of reception and status of reception), and they may each include validation tags for validating references to other secure database records.

Figure 29 shows an example of a detailed format for an administrative event log 442. In the preferred embodiment, administrative event log 442 includes an event log record 442(1) . . . 442(N) for each shipped administrative object and for each received administrative object. Each administrative event log record may include a header 443a and from 1 to N sub-records 442(J)(1) . . . 442(J)(N). In the preferred embodiment, header 443a may include a site record number field 443A(1), a record length field 443A(2), an administrative object ID field 443A(3), a field 443A(4) specifying a number of events, a validation tag 443A(5) from shipping table 444 or receiving table 446, and a check sum field 443A(6). The number of events specified in field 443A(4) corresponds to the number of sub-records 442(J)(1) . . . 442(J)(N) within the administrative event log record 442(J). Each of these sub-records specifies

information about a particular "event" affected or corresponding to the administrative object specified within field 443(A)(3).

Administrative events are retained in the administrative event log 442 to permit the reconstruction (and preparation for
5 construction or processing) of the administrative objects that have been sent from or received by the system. This permits lost administrative objects to be reconstructed at a later time.

Each sub-record may include a sub-record length field
10 442(J)(1)(a), a data area length field 442(J)(1)(b), an event ID field 442(J)(1)(c), a record type field 442(J)(1)(d), a record ID field 442(J)(1)(e), a data area field 442(J)(1)(f), and a check value field 442(J)(1)(g). The data area 442(J)(1)(f) may be used to indicate
15 which information within secure database 610 is affected by the event specified in the event ID field 442(J)(1)(c), or what new secure database item(s) were added, and may also specify the outcome of the event.

The object registration table 460 in the preferred
20 embodiment includes a record corresponding to each VDE object 300 within object storage (repository) 728. When a new object arrives or is detected (e.g., by redirector 684), a preferred embodiment electronic appliance 600 "registers" the object by creating an appropriate object registration record and storing it

in the object registration table 460. In the preferred
embodiment, the object registration table stores information that
is user-independent, and depends only on the objects that are
registered at a given VDE electronic appliance 600. Registration
5 activities are typically managed by a REGISTER method
associated with an object.

In the example, subject table 462 associates users (or
groups of users) with registered objects. The example subject
10 table 462 performs the function of an access control list by
specifying which users are authorized to access which registered
VDE objects 300.

As described above, secure database 610 stores at least one
15 PERC 808 corresponding to each registered VDE object 300.
PERCS 808 specify a set of rights that may be exercised to use or
access the corresponding VDE object 300. The preferred
embodiment allows user to "customize" their access rights by
selecting a subset of rights authorized by a corresponding PERC
20 808 and/or by specifying parameters or choices that correspond
to some or all of the rights granted by PERC 808. These user
choices are set forth in a user rights table 464 in the preferred
embodiment. User rights table (URT) 464 includes URT records,
each of which corresponds to a user (or group of users). Each of

these URT records specifies user choices for a corresponding
VDE object 300. These user choices may, either independently or
in combination with a PERC 808, reference one or more methods
1000 for exercising the rights granted to the user by the PERC
5 808 in a way specified by the choices contained within the URT
record.

Figure 30 shows an example of how these various tables
may interact with one another to provide a secure database
10 lookup mechanism. Figure 30 shows object registration table
460 as having a plurality of object registration records 460(1),
460(2), These records correspond to VDE objects 300(1),
300(2), ... stored within object repository 728. Figure 31 shows
an example format for an object registration record 460 provided
15 by the preferred embodiment. Object registration record 460(N)
may include the following fields:

- site record number field 466(1)
- object type field 466(2)
- creator ID field 466(3)
- 20 object ID field 466(4)
- a reference field 466(5) that references subject
table 462
- an attribute field 466(6)
- a minimum registration interval field 466(7)

a tag 466(8) to a subject table record, and
a check value field 466(9).

5 The site record number field 466(1) specifies the site
record number for this object registration record 460(N). In one
embodiment of secure database 610, each record stored within
the secure database is identified by a site record number. This
site record number may be used as part of a database lookup
process in order to keep track of all of the records within the
10 secure database 610.

Object type field 466(2) may specify the type of registered
VDE object 300 (e.g., a content object, an administrative object,
etc.).

15

Creator ID field 466(3) in the example may identify the
creator of the corresponding VDE object 300.

20

Object ID field 466(4) in the example uniquely identifies
the registered VDE object 300.

Reference field 466(5) in the preferred embodiment
identifies a record within the subject table 462. Through use of
this reference, electronic appliance 600 may determine all users

(or user groups) listed in subject table 462 authorized to access the corresponding VDE object 300. Tag 466(8) is used to validate that the subject table records accessed using field 466(5) is the proper record to be used with the object registration record

5 460(N).

Attribute field 466(6) may store one or more attributes or attribute flags corresponding to VDE object 300.

10 Minimum registration interval field 466(7) may specify how often the end user may re-register as a user of the VDE object 300 with a clearinghouse service, VDE administrator, or VDE provider. One reason to prevent frequent re-registration is to foreclose users from reusing budget quantities in traveling

15 objects until a specified amount of time has elapsed. The minimum registration interval field 466(7) may be left unused when the object owner does not wish to restrict re-registration.

Check value field 466(9) contains validation information

20 used for detecting corruption or modification of record 460(N) to ensure security and integrity of the record. In the preferred embodiment, many or all of the fields within record 460(N) (as with other records within the secure database 610) may be fully or partially encrypted and/or contain fields that are stored

redundantly in each record (once in unencrypted form and once in encrypted form). Encrypted and unencrypted versions of the same fields may be cross checked at various times to detect corruption or modification of the records.

5

As mentioned above, reference field 466(5) references subject table 462, and in particular, references one or more user/object records 460(M) within the subject table. Figure 32 shows an example of a format for a user/object record 462(M) provided by the example. Record 462(M) may include a header 468 and a subject record portion 470. Header 468 may include a field 468(6) referencing a "first" subject record 470 contained within the subject registration table 462. This "first" subject record 470(1) may, in turn, include a reference field 470(5) that references a "next" subject record 470(2) within the subject registration table 462, and so on. This "linked list" structure permits a single object registration record 460(N) to reference to from one to N subject records 470.

20

Subject registration table header 468 in the example includes a site record number field 468(1) that may uniquely identify the header as a record within secure database 610. Header 468 may also include a creator ID field 468(2) that may be a copy of the content of the object registration table creator ID

field 466(3). Similarly, subject registration table header 468 may include an object ID field 468(5) that may be a copy of object ID field 466(4) within object registration table 460. These fields 468(2), 468(5) make user/object registration records explicitly
5 correspond to particular VDE objects 300.

Header 468 may also include a tag 468(7) that permits validation. In one example arrangement, the tag 468(7) within the user/object registration header 468 may be the same as the
10 tag 466(8) within the object registration record 460(N) that points to the user/object registration header. Correspondence between these tags 468(7) and 466(8) permits validation that the object registration record and user/object registration header match up.

15

User/object header 468 also includes an original distributor ID field 468(3) indicating the original distributor of the corresponding VDE object 300, and the last distributor ID field 468(4) that indicates the last distributor within the chain of
20 handling of the object prior to its receipt by electronic appliance 600.

Header 468 also includes a tag 468(8) allowing validation between the header and the "first" subject record 470(1) which field 468(6) references

5 Subject record 470(1) includes a site record number 472(1),
a user (or user group) ID field 472(2), a user (or user group)
attributes field 472(3), a field 472(4) referencing user rights table
464, a field 472(5) that references to the "next" subject record
470(2) (if there is one), a tag 472(6) used to validate with the
10 header tag 468(8), a tag 472(7) used to validate with a
corresponding tag in the user rights table record referenced by
field 472(4), a tag 472(9) used to validate with a tag in the "next"
subject record referenced to by field 472(5) and a check value
field 472(9).

15

 User or user group ID 472(2) identifies a user or a user
group authorized to use the object identified in field 468(5).
Thus, the fields 468(5) and 472(2) together form the heart of the
access control list provided by subject table 462. User attributes
20 field 472(3) may specify attributes pertaining to use/access to
object 300 by the user or user group specified in fields 472(2).
Any number of different users or user groups may be added to
the access control list (each with a different set of attributes

472(3)) by providing additional subject records 470 in the "linked list" structure.

5 Subject record reference field 472(4) references one or more records within user rights table 464. Figure 33 shows an example of a preferred format for a user rights table record 464(k). User rights record 464(k) may include a URT header 474, a record rights header 476, and a set of user choice records 478. URT header 474 may include a site record number field, a
10 field 474(2) specifying the number of rights records within the URT record 464(k), a field 474(3) referencing a "first" rights record (i.e., to rights record header 476), a tag 474(4) used to validate the lookup from the subject table 462, a tag 474(5) used to validate the lookup to the rights record header 476, and a
15 check value field 474(6).

 Rights record header 476 in the preferred embodiment may include site record number field 476(1), a right ID field 476(2), a field 476(3) referencing the "next" rights record 476(2),
20 a field 476(4) referencing a first set of user choice records 478(1), a tag 476(5) to allow validation with URT header tag 474(5), a tag 476(6) to allow validation with a user choice record tag 478(6), and a check value field 476(7). Right ID field 476(2) may, for example, specify the type of right conveyed by the rights

record 476(e.g., right to use, right to distribute, right to read, right to audit, etc.).

5 The one or more user choice records 478 referenced by rights record header 476 sets forth the user choices corresponding to access and/or use of the corresponding VDE object 300. There will typically be a rights record 476 for each right authorized to the corresponding user or user group. These rights govern use of the VDE object 300 by that user or user
10 group. For instance, the user may have an "access" right, and an "extraction" right, but not a "copy" right. Other rights controlled by rights record 476 (which is derived from PERC 808 using a REGISTER method in the preferred embodiment) include distribution rights, audit rights, and pricing rights. When an
15 object 300 is registered with the electronic appliance 600 and is registered with a particular user or user group, the user may be permitted to select among various usage methods set forth in PERC 808. For instance, a VDE object 300 might have two required meter methodologies: one for billing purposes, and one
20 for accumulating data concerning the promotional materials used by the user. The user might be given the choice of a variety of meter/billing methods, such as: payment by VISA or MasterCard; choosing between billing based upon the quantity of material retrieved from an information database, based on the

time of use, and/or both. The user might be offered a discount on time and/or quantity billing if he is willing to allow certain details concerning his retrieval of content to be provided to third parties (e.g., for demographic purposes). At the time of registration of an object and/or user for the object, the user would be asked to select a particular meter methodology as the "active metering method" for the first acquired meter. A VDE distributor might narrow the universe of available choices for the user to a subset of the original selection array stipulated by PERC 808. These user selection and configuration settings are stored within user choice records 480(1), 480(2), 480(N). The user choice records need not be explicitly set forth within user rights table 464; instead, it is possible for user choice records 480 to refer (e.g., by site reference number) to particular VDE methods and/or information parameterizing those methods. Such reference by user choice records 480 to method 1000 should be validated by validation tags contained within the user choice records. Thus, user choice records 480 in the preferred embodiment may select one or more methods 1000 for use with the corresponding VDE object 300 (as is shown in Figure 27). These user choice records 480 may themselves fully define the methods 1000 and other information used to build appropriate components assemblies 690 for implementing the methods. Alternatively, the user/object record 462 used to reference the

user rights record 464 may also reference the PERC 808 corresponding to VDE object 300 to provide additional information needed to build the component assembly 690 and/or otherwise access the VDE object 300. For example, PERC 808
5 may be accessed to obtain MDEs 1202 pertaining to the selected methods, private body and/or rights keys for decrypting and/or encrypting object contents, and may also be used to provide a checking capability ensuring that the user rights record conveys only those rights authorized by a current authorization embodied
10 within a PERC.

In one embodiment provided by the present invention, a conventional database engine may be used to store and organize secure database 610, and the encryption layers discussed above
15 may be "on top of" the conventional database structure. However, if such a conventional database engine is unable to organize the records in secure database 610 and support the security considerations outlined above, then electronic appliance 600 may maintain separate indexing structures in encrypted
20 form. These separate indexing structures can be maintained by SPE 503. This embodiment would require SPE 503 to decrypt the index and search decrypted index blocks to find appropriate "site record IDs" or other pointers. SPE 503 might then request the indicated record from the conventional database engine. If

the record ID cannot be checked against a record list, SPE 503 might be required to ask for the data file itself so it can retrieve the desired record. SPE 503 would then perform appropriate authentication to ensure that the file has not been tampered with and that the proper block is returned. SPE 503 should not simply pass the index to the conventional database engine (unless the database engine is itself secure) since this would allow an incorrect record to be swapped for the requested one.

Figure 34 is an example of how the site record numbers described above may be used to access the various data structures within secure database 610. In this example, secure database 610 further includes a site record table 482 that stores a plurality of site record numbers. Site record table 482 may store what is in effect a "master list" of all records within secure database 610. These site record numbers stored by site record table 482 permit any record within secure database 610 to be accessed. Thus, some of the site records within site record table 482 may index records with an object registration table 460, other site record numbers within the site record table may index records within the user/object table 462, still other site record numbers within the site record table may access records within URT 464, and still other site record numbers within the site record table may access PERCs 808. In addition, each of method

cores 1000' may also include a site record number so they may be accessed by site record table 482.

Figure 34A shows an example of a site record 482(j) within site record table 482. Site record 482(j) may include a field 484(1) indicating the type of record, a field 484(2) indicating the owner or creator of the record, a "class" field 484(3) and an "instance" field 484(4) providing additional information about the record to which the site record 482(j) points; a specific descriptor field 484(5) indicating some specific descriptor (e.g., object ID) associated with the record; an identification 484(6) of the table or other data structure which the site record references; a reference and/or offset within that data structure indicating where the record begins; a validation tag 484(8) for validating the record being looked up, and a check value field 484(9). Fields 484(6) and 484(7) together may provide the mechanism by which the record referenced to by the site record 484(j) is actually physically located within the secure database 610.

20 Updating Secure Database 610

Figure 35 show an example of a process 1150 which can be used by a clearinghouse, VDE administrator or other VDE participant to update the secure database 610 maintained by an end user's electronic appliance 600. For example, the process

1500 shown in Figure 35 might be used to collect "audit trail"

records within secure database 610 and/or provide new budgets and permissions (e.g., PERCs 808) in response to an end user's request.

5

Typically, the end user's electronic appliance 600 may initiate communications with a clearinghouse (Block 1152). This contact may, for example, be established automatically or in response to a user command. It may be initiated across the

10

electronic highway 108, or across other communications networks such as a LAN, WAN, two-way cable or using portable media exchange between electronic appliances. The process of exchanging administrative information need not occur in a single "on line" session, but could instead occur over time based on a

15

number of different one-way and/or two-way communications over the same or different communications means. However, the process 1150 shown in Figure 35 is a specific example where the end user's electronic appliance 600 and the other VDE

20

participant (e.g., a clearinghouse) establish a two-way real-time interactive communications exchange across a telephone line, network, electronic highway 108, etc.

The end user's electronic appliance 600 generally contacts a particular VDE administrator or clearinghouse. The identity of

the particular clearinghouse is based on the VDE object 300 the user wishes to access or has already accessed. For example, suppose the user has already accessed a particular VDE object 300 and has run out of budget for further access. The user could
5 issue a request which will cause her electronic appliance 600 to automatically contact the VDE administrator, distributor and/or financial clearinghouse that has responsibility for that particular object. The identity of the appropriate VDE participants to contact is provided in the example by information within UDEs
10 1200, MDEs 1202, the Object Registration Table 460 and/or Subject Table 462, for example. Electronic appliance 600 may have to contact multiple VDE participants (e.g., to distribute audit records to one participant, obtain additional budgets or other permissions from another participant. etc.). The contact
15 1152 may in one example be scheduled in accordance with the Figure 27 Shipping Table 444 and the Figure 29 Administrative Event Log 442.

Once contact is established, the end user's electronic
20 appliance and the clearinghouse typically authenticate one another and agree on a session key to use for the real-time information exchange (Block 1154). Once a secure connection is established, the end user's electronic appliance may determine (e.g., based on Shipping Table 444) whether it has any

administrative object(s) containing audit information that it is supposed to send to the clearinghouse (decision Block 1156).

Audit information pertaining to several VDE objects 300 may be placed within the same administrative object for transmission, or
5 different administrative objects may contain audit information about different objects. Assuming the end user's electronic appliance has at least one such administrative object to send to this particular clearinghouse ("yes" exit to decision Block 1156), the electronic appliance sends that administrative object to the
10 clearinghouse via the now-established secure real-time communications (Block 1158). In one specific example, a single administrative object may be sent an administrative object containing audit information pertaining to multiple VDE objects, with the audit information for each different object
15 compromising a separate "event" within the administrative object.

The clearinghouse may receive the administrative object and process its contents to determine whether the contents are
20 "valid" and "legitimate." For example, the clearinghouse may analyze the contained audit information to determine whether it indicates misuse of the applicable VDE object 300. The clearinghouse may, as a result of this analysis, may generate one or more responsive administrative objects that it then sends to

the end user's electronic appliance 600 (Block 1160). The end user's electronic appliance 600 may process events that update its secure database 610 and/or SPU 500 contents based on the administrative object received (Block 1162). For example, if the
5 audit information received by the clearinghouse is legitimate, then the clearinghouse may send an administrative object to the end user's electronic appliance 600 requesting the electronic appliance to delete and/or compress the audit information that has been transferred. Alternatively or in addition, the
10 clearinghouse may request additional information from the end-user electronic appliance 600 at this stage (e.g., retransmission of certain information that was corrupted during the initial transmission, transmission of additional information not earlier transmitted, etc.). If the clearinghouse detects misuse based on
15 the received audit information, it may transmit an administrative object that revokes or otherwise modifies the end user's right to further access the associated VDE objects 300.

The clearinghouse may, in addition or alternatively, send
20 an administrative object to the end user's electronic appliance 600 that instructs the electronic appliance to display one or more messages to the user. These messages may inform the user about certain conditions and/or they may request additional information from the user. For example, the message may

instruct the end user to contact the clearinghouse directly by telephone or otherwise to resolve an indicated problem, enter a PIN, or it may instruct the user to contact a new service company to re-register the associated VDE object. Alternatively,
5 the message may tell the end user that she needs to acquire new usage permissions for the object, and may inform the user of cost, status and other associated information.

During the same or different communications exchange,
10 the same or different clearinghouse may handle the end user's request for additional budget and/or permission pertaining to VDE object 300. For example, the end user's electronic appliance 600 may (e.g., in response to a user input request to access a particular VDE object 300) send an administrative object to the
15 clearinghouse requesting budgets and/or other permissions allowing access (Block 1164). As mentioned above, such requests may be transmitted in the form of one or more administrative objects, such as, for example, a single administrative object having multiple "events" associated with multiple requested
20 budgets and/or other permissions for the same or different VDE objects 300. The clearinghouse may upon receipt of such a request, check the end user's credit, financial records, business agreements and/or audit histories to determine whether the requested budgets and/or permissions should be given. The

clearinghouse may, based on this analysis, send one or more responsive administrative objects which cause the end user's electronic appliance 600 to update its secure database in response (Block 1166, 1168). This updating might, for example, comprise replacing an expired PERC 808 with a fresh one, modifying a PERC to provide additional (or lesser) rights, etc. Steps 1164-1168 may be repeated multiple times in the same or different communications session to provide further updates to the end user's secure database 610.

Figure 36 shows an example of how a new record or element may be inserted into secure database 610. The load process 1070 shown in Figure 35 checks each data element or item as it is loaded to ensure that it has not been tampered with, replaced or substituted. In the process 1070 shown in Figure 35, the first step that is performed is to check to see if the current user of electronic appliance 600 is authorized to insert the item into secure database 610 (block 1072). This test may involve, in the preferred embodiment, loading (or using already loaded) appropriate methods 1000 and other data structures such as UDEs 1200 into an SPE 503, which then authenticates user authorization to make the change to secure database 610 (block 1074). If the user is approved as being authorized to make the change to secure database 610, then SPE 503 may check the

integrity of the element to be added to the secure database by decrypting it (block 1076) and determining whether it has become damaged or corrupted (block 1078). The element is checked to ensure that it decrypts properly using a

5 predetermined management file key, and the check value may be validated. In addition, the public and private header ID tags (if present) may be compared to ensure that the proper element has been provided and had not been substituted, and the unique element tag ID compared against the predetermined element

10 tag. If any of these tests fail, the element may be automatically rejected, error corrected, etc. Assuming the element is found to have integrity, SPE 503 may re-encrypt the information (block 1080) using a new key for example (see Figure 37 discussion below). In the same process step an appropriate tag is preferably

15 provided so that the information becomes encrypted within a security wrapper having appropriate tags contained therein (block 1082). SPE 503 may retain appropriate tag information so that it can later validate or otherwise authenticate the item when it is again read from secure database 610 (block 1084).

20 The now-secure element within its security wrapper may then be stored within secure database 610.

Figure 37 shows an example of a process 1050 used in the preferred embodiment database to securely access an item stored

in secure database 610. In the preferred embodiment, SPE 503 first accesses and reads in the item from secure database 610 records. SPE 503 reads this information from secure database 610 in encrypted form, and may "unwrap" it (block 1052) by
5 decrypting it (block 1053) based on access keys internally stored within the protected memory of an SPU 500. In the preferred embodiment, this "unwrap" process 1052 involves sending blocks of information to encrypt/decrypt engine 522 along with a management file key and other necessary information needed to
10 decrypt. Decrypt engine 522 may return "plaintext" information that SPE 503 then checks to ensure that the security of the object has not been breached and that the object is the proper object to be used (block 1054). SPE 503 may then check all correlation and access tags to ensure that the read-in element
15 has not been substituted and to guard against other security threats (block 1054). Part of this "checking" process involves checking the tags obtained from the secure database 610 with tags contained within the secure memory or an SPU 500 (block 1056). These tags stored within SPU 500 may be accessed from
20 SPU protected memory (block 1056) and used to check further the now-unwrapped object. Assuming this "checking" process 1054 does not reveal any improprieties (and block 1052 also indicates that the object has not become corrupted or otherwise damaged), SPE 503 may then access or otherwise use the item

(block 1058). Once use of the item is completed, SPE 503 may need to store the item back into secure database 610 if it has changed. If the item has changed, SPE 503 will send the item in its changed form to encrypt/decrypt engine 522 for encryption (block 1060), providing the appropriate necessary information to the encrypt/decrypt engine (e.g., the appropriate same or different management file key and data) so that the object is appropriately encrypted. A unique, new tag and/or encryption key may be used at this stage to uniquely tag and/or encrypt the item security wrapper (block 1062; see also detailed Figure 37 discussion below). SPE 503 may retain a copy of the key and/or tag within a protected memory of SPU 500 (block 1064) so that the SPE can decrypt and validate the object when it is again read from secure database 610.

15

The keys to decrypt secure database 610 records are, in the preferred embodiment, maintained solely within the protected memory of an SPU 500. Each index or record update that leaves the SPU 500 may be time stamped, and then encrypted with a unique key that is determined by the SPE 503. For example, a key identification number may be placed "in plain view" at the front of the records of secure database 610 so the SPE 503 can determine which key to use the next time the record is retrieved. SPE 503 can maintain the site ID of the record or index, the key

20

identification number associated with it, and the actual keys in the list internal to the SPE. At some point, this internal list may fill up. At this point, SPE 503 may call a maintenance routine that re-encrypts items within secure database 610 containing changed information. Some or all of the items within the data structure containing changed information may be read in, decrypted, and then re-encrypted with the same key. These items may then be issued the same key identification number. The items may then be written out of SPE 503 back into secure database 610. SPE 503 may then clear the internal list of item IDs and corresponding key identification numbers. It may then begin again the process of assigning a different key and a new key identification number to each new or changed item. By using this process, SPE 503 can protect the data structures (including the indexes) of secure database 610 against substitution of old items and against substitution of indexes for current items. This process also allows SPE 503 to validate retrieved item IDs against the encrypted list of expected IDs.

Figure 38 is a flowchart showing this process in more detail. Whenever a secure database 610 item is updated or modified, a new encryption key can be generated for the updated item. Encryption using a new key is performed to add security and to prevent misuse of backup copies of secure database 610

records. The new encryption key for each updated secure database 610 record may be stored in SPU 500 secure memory with an indication of the secure database record or record(s) to which it applies.

5

SPE 503 may generate a new encryption/decryption key for each new item it is going to store within secure database 610 (block 1086). SPE 503 may use this new key to encrypt the record prior to storing it in the secure database (block 1088).

10

SPE 503 make sure that it retains the key so that it can later read and decrypt the record. Such decryption keys are, in the preferred embodiment, maintained within protected non-volatile memory (e.g., NVRAM 534b) within SPU 500. Since this protected memory has a limited size, there may not be enough

15

room within the protected memory to store a new key. This condition is tested for by decision block 1090 in the preferred embodiment. If there is not enough room in memory for the new key (or some other event such as the number of keys stored in the memory exceeding a predetermined number, a timer has expired, etc.), then the preferred embodiment handles the situation by re-encrypting other records with secure database 610 with the same new key in order to reduce the number of (or change) encryption/decryption keys in use. Thus, one or more secure database 610 items may be read from the secure database

20

(block 1092), and decrypted using the old key(s) used to encrypt them the last time they were stored. In the preferred embodiment, one or more "old keys" are selected, and all secure database items encrypted using the old key(s) are read and
5 decrypted. These records may now be re-encrypted using the new key that was generated at block 1086 for the new record (block 1094). The old key(s) used to decrypt the other record(s) may now be removed from the SPU protected memory (block 1096), and the new key stored in its place (block 1097). The old
10 key(s) cannot be removed from secure memory by block 1096 unless SPE 503 is assured that all records within the secure database 610 that were encrypted using the old key(s) have been read by block 1092 and re-encrypted by block 1904 using the new key. All records encrypted (or re-encrypted) using the new key
15 may now be stored in secure database 610 (block 1098). If decision block 1090 determines there is room within the SPU 500 protected memory to store the new key, then the operations of blocks 1092, 1094, 1096 are not needed and SPE 503 may instead simply store the new key within the protected memory
20 (block 1097) and store the new encrypted records into secure database 610 (block 1098).

The security of secure database 610 files may be further improved by segmenting the records into "compartments."

Different encryption/decryption keys may be used to protect different "compartments." This strategy can be used to limit the amount of information within secure database 610 that is encrypted with a single key. Another technique for increasing security of secure database 610 may be to encrypt different portions of the same records with different keys so that more than one key may be needed to decrypt those records.

Backup of Secure Database 610

Secure database 610 in the preferred embodiment is backed up at periodic or other time intervals to protect the information the secure database contains. This secure database information may be of substantial value to many VDE participants. Back ups of secure database 610 should occur without significant inconvenience to the user, and should not breach any security.

The need to back up secure database 610 may be checked at power on of electronic appliance 600, when SPE 503 is initially invoked, at periodic time intervals, and if "audit roll up" value or other summary services information maintained by SPE 503 exceeds a user set or other threshold, or triggered by criteria established by one or more content publishers and/or distributors and/or clearinghouse service providers and/or users. The user

may be prompted to backup if she has failed to do so by or at some certain point in time or after a certain duration of time or quantity of usage, or the backup may proceed automatically without user intervention.

5

Referring to Figure 8, backup storage 668 and storage media 670 (e.g., magnetic tape) may be used to store backed up information. Of course, any non-volatile media (e.g., one or more floppy diskettes, a writable optical diskette, a hard drive, or the like) may be used for backup storage 668.

10

There are at least two scenarios to backing up secure database 610. The first scenario is "site specific," and uses the security of SPU 500 to support restoration of the backed up information. This first method is used in case of damage to secure database 610 due for example to failure of secondary storage device 652, inadvertent user damage to the files, or other occurrences that may damage or corrupt some or all of secure database 610. This first, site specific scenario of back up assumes that an SPU 500 still functions properly and is available to restore backed up information.

15

20

The second back up scenario assumes that the user's SPU 500 is no longer operational and needs to be, or has been,

replaced. This second approach permits an authorized VDE administrator or other authorized VDE participant to access the stored back up information in order to prevent loss of critical data and/or assist the user in recovering from the error.

5

Both of these scenarios are provided by the example of program control steps performed by ROS 602 shown in Figure 39. Figure 39 shows an example back up routine 1250 performed by an electronic appliance 600 to back up secure database 610 (and other information) onto back up storage 668. Once a back up has been initiated, as discussed above, back up routine 1250 generates one or more back up keys (block 1252). Back up routine 1250 then reads all secure database items, decrypts each item using the original key used to encrypt them before they were stored in secure database 610 (block 1254). Since SPU 500 is typically the only place where the keys for decrypting this information within an instance of secure database 610 are stored, and since one of the scenarios provided by back up routine 1250 is that SPU 500 completely failed or is destroyed, back up routine 1250 performs this reading and decrypting step 1254 so that recovery from a backup is not dependent on knowledge of these keys within the SPU. Instead, back up routine 1250 encrypts each secure database 610 item with a newly generated back up key(s) (block 1256) and writes the

encrypted item to back up store 668 (block 1258). This process continues until all items within secure database 610 have been read, decrypted, encrypted with a newly generated back up key(s), and written to the back up store (as tested for by decision
5 block 1260).

The preferred embodiment also reads the summary services audit information stored within the protected memory of SPU 500 by SPE summary services manager 560, encrypts this
10 information with the newly generated back up key(s), and writes this summary services information to back up store 668 (block 1262).

Finally, back up routine 1250 saves the back up key(s)
15 generated by block 1252 and used to encrypt in blocks 1256, 1262 onto back up store 668. It does this in two secure ways in order to cover both of the restoration scenarios discussed above. Back up routine 1250 may encrypt the back up key(s) (along with other information such as the time of back up and other
20 appropriate information to identify the back up) with a further key or keys such that only SPU 500 can decrypt (block 1264). This encrypted information is then written to back up store 668 (block 1264). For example, this step may include multiple encryptions using one or more public keys with corresponding

private keys known only to SPU 500. Alternatively, a second
back up key generated by the SPU 500 and kept only in the SPU
may be used for the final encryption in place of a public key.
Block 1264 preferably includes multiple encryption in order to
5 make it more difficult to attack the security of the back up by
"cracking" the encryption used to protect the back up keys.
Although block 1262 includes encrypted summary services
information on the back up, it preferably does not include SPU
device private keys, shared keys, SPU code and other internal
10 security information to prevent this information from ever
becoming available to users even in encrypted form.

The information stored by block 1264 is sufficient to allow
the same SPU 500 that performed (or at least in part performed)
15 back up routine 1250 to recover the backed up information.
However, this information is useless to any device other than
that same SPU because only that SPU knows the particular keys
used to protect the back up keys. To cover the other possible
scenario wherein the SPU 500 fails in a non-recoverable way,
20 back up routine 1250 provides an additional step (block 1266) of
saving the back up key(s) under protection of one or more further
set of keys that may be read by an authorized VDE
administrator. For example, block 1266 may encrypt the back up
keys with an "download authorization key" received during

initialization of SPU 500 from a VDE administrator. This encrypted version of back up keys is also written to back up store 668 (block 1266). It can be used to support restoration of the back up files in the event of an SPU 500 failure. More specifically, a VDE administrator that knows the download authorization (or other) keys(s) used by block 1266 may be able to recover the back up key(s) in the back up store 668 and proceed to restore the backed up secure database 610 to the same or different electronic appliance 600.

In the preferred embodiment, the information saved by routine 1250 in back up files can be restored only after receiving a back up authorization from an authorized VDE administrator. In most cases, the restoration process will simply be a restoration of secure database 610 with some adjustments to account for any usage since the back up occurred. This may require the user to contact additional providers to transmit audit and billing data and receive new budgets to reflect activity since the last back up. Current summary services information maintained within SPU 500 may be compared to the summary services information stored on the back up to determine or estimate most recent usage activity.

In case of an SPU 500 failure, an authorized VDE administrator must be contacted to both initialize the replacement SPU 500 and to decrypt the back up files. These processes allow for both SPU failures and upgrades to new SPUs.

5 In the case of restoration, the back up files are used to restore the necessary information to the user's system. In the case of upgrades, the back up files may be used to validate the upgrade process.

10 The back up files may in some instances be used to transfer management information between electronic appliances 600. However, the preferred embodiment may restrict some or all information from being transportable between electronic appliances with appropriate authorizations. Some or all of the
15 back up files may be packaged within an administrative object and transmitted for analysis, transportation, or other uses.

20 As a more detailed example of a need for restoration from back up files, suppose an electronic appliance 600 suffers a hard disk failure or other accident that wipes out or corrupts part or all of the secure database 610, but assume that the SPU 500 is still functional. SPU 500 may include all of the information (e.g., secret keys and the like) it needs to restore the secure database 610. However, ROS 602 may prevent secure database

restoration until a restoration authorization is received from a VDE administrator. A restoration authorization may comprise, for example, a "secret value" that must match a value expected by SPE 503. A VDE administrator may, if desired, only provide
5 this restoration authorization after, for example, summary services information stored within SPU 500 is transmitted to the administrator in an administrative object for analysis. In some circumstances, a VDE administrator may require that a copy (partial or complete) of the back up files be transmitted to it
10 within an administrative object to check for indications of fraudulent activities by the user. The restoration process, once authorized, may require adjustment of restored budget records and the like to reflect activity since the last back up, as mentioned above.

15

Figure 40 is an example of program controlled "restore" routine 1268 performed by electronic appliance 600 to restore secure database 610 based on the back up provided by the routine shown in Figure 38. This restore may be used, for
20 example, in the event that an electronic appliance 600 has failed but can be recovered or "reinitialized" through contact with a VDE administrator for example. Since the preferred embodiment does not permit an SPU 500 to restore from backup unless and until authorized by a VDE administrator, restore

routine 1268 begins by establishing a secure communication with a VDE administrator that can authorize the restore to occur (block 1270). Once SPU 500 and the VDE administrator authenticate one another (part of block 1270), the VDE administrator may extract "work in progress" and summary values from the SPU 500's internal non-volatile memory (block 1272). The VDE administrator may use this extracted information to help determine, for example, whether there has been a security violation, and also permits a failed SPU 500 to effectively "dump" its contents to the VDE administrator to permit the VDE administrator to handle the contents. The SPU 500 may encrypt this information and provide it to the VDE administrator packaged in one or more administrative objects. The VDE administrator may then request a copy of some or all of the current backup of secure database 610 from the SPU 500 (block 1274). This information may be packaged by SPU 500 into one or more administrative objects, for example, and sent to the VDE administrator. Upon receiving the information, the VDE administrator may read the summary services audit information from the backup volume (i.e., information stored by Figure 38 block 1262) to determine the summary values and other information stored at time of backup. The VDE administrator may also determine the time and date the backup was made by reading the information stored by Figure 38 block 1264.

The VDE administrator may at this point restore the summary values and other information within SPU 500 based on the information obtained by block 1272 and from the backup (block 1276). For example, the VDE administrator may reset
5 SPU internal summary values and counters so that they are consistent with the last backup. These values may be adjusted by the VDE administrator based on the "work in progress" recovered by block 1272, the amount of time that has passed since the backup, etc. The goal may typically be to attempt to
10 provide internal SPU values that are equal to what they would have been had the failure not occurred.

The VDE administrator may then authorize SPU 500 to recover its secure database 610 from the backup files (block
15 1278). This restoration process replaces all secure database 610 records with the records from the backup. The VDE administrator may adjust these records as needed by passing commands to SPU 500 during or after the restoration process.

20 The VDE administrator may then compute bills based on the recovered values (block 1280), and perform other actions to recover from SPU downtime (block 1282). Typically, the goal is to bill the user and adjust other VDE 100 values pertaining to the failed electronic appliance 600 for usage that occurred

subsequent to the last backup but prior to the failure. This process may involve the VDE administrator obtaining, from other VDE participants, reports and other information pertaining to usage by the electronic appliance prior to its failure and comparing it to the secure database backup to determine which usage and other events are not yet accounted for.

In one alternate embodiment, SPU 500 may have sufficient internal, non-volatile memory to allow it to store some or all of secure database 610. In this embodiment, the additional memory may be provided by additional one or more integrated circuits that can be contained within a secure enclosure, such as a tamper resistant metal container or some form of a chip pack containing multiple integrated circuit components, and which impedes and/or evidences tampering attempts, and/or disables a portion or all of SPU 500 or associated critical key and/or other control information in the event of tampering. The same back up routine 1250 shown in Figure 38 may be used to back up this type of information, the only difference being that block 1254 may read the secure database item from the SPU internal memory and may not need to decrypt it before encrypting it with the back up key(s).

Event-Driven VDE Processes

As discussed above, processes provided by/under the preferred embodiment rights operating system (ROS) 602 may be "event driven." This "event driven" capability facilitates
5 integration and extendibility.

An "event" is a happening at a point in time. Some examples of "events" are a user striking a key of a keyboard, arrival of a message or an object 300, expiration of a timer, or a
10 request from another process.

In the preferred embodiment, ROS 602 responds to an "event" by performing a process in response to the event. ROS 602 dynamically creates active processes and tasks in response
15 to the occurrence of an event. For example, ROS 602 may create and begin executing one or more component assemblies 690 for performing a process or processes in response to occurrence of an event. The active processes and tasks may terminate once ROS 602 has responded to the event. This ability to dynamically
20 create (and end) tasks in response to events provides great flexibility, and also permits limited execution resources such as those provided by an SPU 500 to perform a virtually unlimited variety of different processes in different contexts.

Since an "event" may be any type of happening, there are an unlimited number of different events. Thus, any attempt to categorize events into different types will necessarily be a generalization. Keeping this in mind, it is possible to categorize events provided/supported by the preferred embodiment into two broad categories:

- user-initiated events; and
- system-initiated events.

10

Generally, "user-initiated" events are happenings attributable to a user (or a user application). A common "user-initiated" event is a user's request (e.g., by pushing a keyboard button, or transparently using redirector 684) to access an object 300 or other VDE-protected information.

15

"System-initiated" events are generally happenings not attributable to a user. Examples of system initiated events include the expiration of a timer indicating that information should be backed to non-volatile memory, receipt of a message from another electronic appliance 600, and a service call generated by another process (which may have been started to respond to a system-initiated event and/or a user-initiated event).

20

ROS 602 provided by the preferred embodiment responds to an event by specifying and beginning processes to process the event. These processes are, in the preferred embodiment, based on methods 1000. Since there are an unlimited number of
5 different types of events, the preferred embodiment supports an unlimited number of different processes to process events. This flexibility is supported by the dynamic creation of component assemblies 690 from independently deliverable modules such as method cores 1000', load modules 1100, and data structures such
10 as UDEs 1200. Even though any categorization of the unlimited potential types of processes supported/provided by the preferred embodiment will be a generalization, it is possible to generally classify processes as falling within two categories:

- 15 • processes relating to use of VDE protected information;
and
- processes relating to VDE administration.

"Use" and "Administrative" Processes

20 "Use" processes relate in some way to use of VDE-protected information. Methods 1000 provided by the preferred embodiment may provide processes for creating and maintaining a chain of control for use of VDE-protected information. One specific example of a "use" type process is processing to permit a

user to open a VDE object 300 and access its contents. A method 1000 may provide detailed use-related processes such as, for example, releasing content to the user as requested (if permitted), and updating meters, budgets, audit trails, etc. Use-related processes are often user-initiated, but some use processes may be system-initiated. Events that trigger a VDE use-related process may be called "use events."

An "administrative" process helps to keep VDE 100 working. It provides processing that helps support the transaction management "infrastructure" that keeps VDE 100 running securely and efficiently. Administrative processes may, for example, provide processing relating to some aspect of creating, modifying and/or destroying VDE-protected data structures that establish and maintain VDE's chain of handling and control. For example, "administrative" processes may store, update, modify or destroy information contained within a VDE electronic appliance 600 secure database 610. Administrative processes also may provide communications services that establish, maintain and support secure communications between different VDE electronic appliances 600. Events that trigger administrative processes may be called "administrative events."

Reciprocal Methods

Some VDE processes are paired based on the way they interact together. One VDE process may "request" processing services from another VDE process. The process that requests processing services may be called a "request process." The "request" constitutes an "event" because it triggers processing by the other VDE process in the pair. The VDE process that responds to the "request event" may be called a "response process." The "request process" and "response process" may be called "reciprocal processes."

The "request event" may comprise, for example, a message issued by one VDE node electronic appliance 600 or process for certain information. A corresponding "response process" may respond to the "request event" by, for example, sending the information requested in the message. This response may itself constitute a "request event" if it triggers a further VDE "response process." For example, receipt of a message in response to an earlier-generated request may trigger a "reply process." This "reply process" is a special type of "response process" that is triggered in response to a "reply" from another "response process." There may be any number of "request" and "response" process pairs within a given VDE transaction.

A "request process" and its paired "response process" may be performed on the same VDE electronic appliance 600, or the two processes may be performed on different VDE electronic appliances. Communication between the two processes in the pair may be by way of a secure (VDE-protected) communication, an "out of channel" communication, or a combination of the two.

Figures 41a-41d are a set of examples that show how the chain of handling and control is enabled using "reciprocal methods." A chain of handling and control is constructed, in part, using one or more pairs of "reciprocal events" that cooperate in request-response manner. Pairs of reciprocal events may be managed in the preferred embodiment in one or more "reciprocal methods." As mentioned above, a "reciprocal method" is a method 1000 that can respond to one or more "reciprocal events." Reciprocal methods contain the two halves of a cooperative process that may be securely executed at physically and/or temporally distant VDE nodes. The reciprocal processes may have a flexibly defined information passing protocols and information content structure. The reciprocal methods may, in fact, be based on the same or different method core 1000' operating in the same or different VDE nodes 600. VDE nodes 600A and 600B shown in Figure 41a may be the same physical

electronic appliance 600 or may be separate electronic appliances.

Figure 41a is an example of the operation of a single pair of reciprocal events. In VDE node 600A, method 1000a is processing an event that has a request that needs to be processed at VDE node 600B. The method 1000a (e.g., based on a component assembly 690 including its associated load modules 1100 and data) that responds to this "request" event is shown in Figure 41a as 1450. The process 1450 creates a request (1452) and, optionally, some information or data that will be sent to the other VDE node 1000b for processing by a process associated with the reciprocal event. The request and other information may be transmitted by any of the transport mechanisms described elsewhere in this disclosure.

Receipt of the request by VDE node 600b comprises a response event at that node. Upon receipt of the request, the VDE node 600b may perform a "reciprocal" process 1454 defined by the same or different method 1000b to respond to the response event. The reciprocal process 1454 may be based on a component assembly 690 (e.g., one or more load modules 1100, data, and optionally other methods present in the VDE node 600B).

Figure 41b extends the concepts presented in Figure 41a to include a response from VDE node 600B back to VDE node 600A. The process starts as described for Figure 41a through the receipt and processing of the request event and information 1452 by the response process 1454 in VDE node 600B. The response process 1454 may, as part of its processing, cooperate with another request process (1468) to send a response 1469 back to the initiating VDE node 600A. A corresponding reciprocal process 1470 provided by method 1000A may respond to and process this request event 1469. In this manner, two or more VDE nodes 600A, 600B may cooperate and pass configurable information and requests between methods 1000A, 1000B executing in the nodes. The first and second request-response sequences [(1450, 1452, 1454) and (1468, 1469, 1470)] may be separated by temporal and spatial distances. For efficiency, the request (1468) and response (1454) processes may be based on the same method 1000 or they may be implemented as two methods in the same or different method core 1000'. A method 1000 may be parameterized by an "event code" so it may provide different behaviors/results for different events, or different methods may be provided for different events.

Figure 41c shows the extension the control mechanism described in Figures 41a-41b to three nodes (600A, 600B, 600C).

Each request-response pair operates in the manner as described for Figure 41b, with several pairs linked together to form a chain of control and handling between several VDE nodes 600A, 600B, 600C. This mechanism may be used to extend the chain of
5 handling and control to an arbitrary number of VDE nodes using any configuration of nodes. For example, VDE node 600C might communicate directly to VDE node 600A and communicate directly to VDE 600B, which in turn communicates with VDE node 600A. Alternately, VDE node 600C might communicate
10 directly with VDE node 600A, VDE node 600A may communicate with VDE node 600B, and VDE node 600B may communicate with VDE node 600C.

A method 1000 may be parameterized with sets of events
15 that specify related or cooperative functions. Events may be logically grouped by function (e.g., use, distribute), or a set of reciprocal events that specify processes that may operate in conjunction with each other. Figure 41d illustrates a set of "reciprocal events" that support cooperative processing between
20 several VDE nodes 102, 106, 112 in a content distribution model to support the distribution of budget. The chain of handling and control, in this example, is enabled by using a set of "reciprocal events" specified within a BUDGET method. Figure 41d is an example of how the reciprocal event behavior within an example

BUDGET method (1510) work in cooperation to establish a chain of handling and control between several VDE nodes. The example BUDGET method 1510 responds to a "use" event 1478 by performing a "use" process 1476 that defines the mechanism by which processes are budgeted. The BUDGET method 1510 might, for example, specify a use process 1476 that compares a meter count to a budget value and fail the operation if the meter count exceeds the budget value. It might also write an audit trail that describes the results of said BUDGET decisions.

Budget method 1510 may respond to a "distribute" event by performing a distribute process 1472 that defines the process and/or control information for further distribution of the budget. It may respond to a "request" event 1480 by performing a request process 1480 that specifies how the user might request use and/or distribution rights from a distributor. It may respond to a "response" event 1482 by performing a response process 1484 that specifies the manner in which a distributor would respond to requests from other users to whom they have distributed some (or all) of their budget to. It may respond to a "reply" event 1474 by performing a reply process 1475 that might specify how the user should respond to message regranting or denying (more) budget.

Control of event processing, reciprocal events, and their associated methods and method components is provided by PERCs 808 in the preferred embodiment. These PERCs (808) might reference administrative methods that govern the creation, modification, and distribution of the data structures and administrative methods that permit access, modification, and further distribution of these items. In this way, each link in the chain of handling and control might, for example, be able to customize audit information, alter the budget requirements for using the content, and/or control further distribution of these rights in a manner specified by prior members along the distribution chain.

In the example shown in Figure 41d, a distributor at a VDE distributor node (106) might request budget from a content creator at another node (102). This request may be made in the context of a secure VDE communication or it may be passed in an "out-of-channel" communication (e.g. a telephone call or letter). The creator 102 may decide to grant budget to the distributor 106 and processes a distribute event (1452 in BUDGET method 1510 at VDE node 102). A result of processing the distribute event within the BUDGET method might be a secure communication (1454) between VDE nodes 102 and 106 by which a budget granting use and redistribute rights to the

distributor 106 may be transferred from the creator 102 to the distributor. The distributor's VDE node 106 may respond to the receipt of the budget information by processing the communication using the reply process 1475B of the BUDGET method 1510. The reply event processing 1475B might, for example, install a budget and PERC 808 within the distributor's VDE 106 node to permit the distributor to access content or processes for which access is control at least in part by the budget and/or PERC. At some point, the distributor 106 may also desire to use the content to which she has been granted rights to access.

After registering to use the content object, the user 112 would be required to utilize an array of "use" processes 1476C to, for example, open, read, write, and/or close the content object as part of the use process.

Once the distributor 106 has used some or all of her budget, she may desire to obtain additional budget. The distributor 106 might then initiate a process using the BUDGET method request process (1480B). Request process 1480B might initiate a communication (1482AB) with the content creator VDE node 102 requesting more budget and perhaps providing details of the use activity to date (e.g., audit trails). The content creator

102 processes the 'get more budget' request event 1482AB using the response process (1484A) within the creator's BUDGET method 1510A. Response process 1484A might, for example, make a determination if the use information indicates proper use of the content, and/or if the distributor is credit worthy for more budget. The BUDGET method response process 1484A might also initiate a financial transaction to transfer funds from the distributor to pay for said use, or use the distribute process 1472A to distribute budget to the distributor 106. A response to the distributor 106 granting more budget (or denying more budget) might be sent immediately as a response to the request communication 1482AB, or it might be sent at a later time as part of a separate communication. The response communication, upon being received at the distributor's VDE node 106, might be processed using the reply process 1475B within the distributor's copy of the BUDGET method 1510B. The reply process 1475B might then process the additional budget in the same manner as described above.

20 The chain of handling and control may, in addition to posting budget information, also pass control information that governs the manner in which said budget may be utilized. For example, the control information specified in the above example may also contain control information describing the process and

limits that apply to the distributor's redistribution of the right to use the creator's content object. Thus, when the distributor responds to a budget request from a user (a communication between a user at VDE node 112 to the distributor at VDE node 106 similar in nature to the one described above between VDE nodes 106 and 102) using the distribute process 1472B within the distributor's copy of the BUDGET method 1510B, a distribution and request/response/reply process similar to the one described above might be initiated.

Thus, in this example a single method can provide multiple dynamic behaviors based on different "triggering" events. For example, single BUDGET method 1510 might support any or all of the events listed below:

Event Type	Event	Process Description
"Use" Events	use budget	Use budget.
Request Events	request more budget	Request more money for budget.
Processed by		
User Node	request audit by auditor #1	Request that auditor #1 audit the budget use.
Request Process	request budget deletion	Request that budget be deleted from system.
1480c	request method updated	Update method used for auditing.
	request to change auditors	Change from auditor 1 to auditor 2, or vice versa.
	request different audit interval	Change time interval between audits.
	request ability to provide budget copies	Request ability to provide copies of a budget.

Event Type	Event	Process Description
5 Response Events Processed by User Node Request Process 1480C	request ability to distribute budget	Request ability to distribute a budget to other users.
	request account status	Request information on current status of an account.
	Request New Method	Request new method.
	Request Method Update	Request update of method.
	Request Method Deletion	Request deletion of method.
	receive more budget	Allocate more money to budget.
	receive method update	Update method.
	receive auditor change	Change from one auditor to another.
	receive change to audit interval	Change interval between audits.
	receive budget deletion	Delete budget.
	provide audit to auditor #1	Forward audit information to auditor #1.
	provide audit to auditor #2	Forward audit information to auditor #2.
	receive account status	Provide account status.
	Receive New	Receive new budget.
	Receive Method Update	Receive updated information.
10 "Distribute" Events	Receive More	Receive more for budget.
	Sent Audit	Send audit information.
	Perform Deletion	Delete information.
	Create New	Create new budget.
	Provide More	Provide more for budget.
	Audit	Perform audit.
	Delete	Delete information.
	Reconcile	Reconcile budget and auditing.
	Copy	Copy budget.
	Distribute	Distribute budget.
15 "Request" Events Processed by Distributor Node Request Process 1484B	Method Modification	Modify method.
	Display Method	Display requested method.
	Delete	Delete information.
	Get New	Get new budget.
	Get More	Get more for budget.
	Get Updated	Get updated information.
	Get Audited	Get audit information.

Event Type	Event	Process Description
Response Events" Processed by Distributor Node Request Process 1484B	Provide New to user	Provide new budget to user.
	Provide More to user	Provide more budget to user.
	Provide Update to user	Provided updated budget to user.
	Audit user	Audit a specified user.
	Delete user's method	Delete method belonging to user.

Examples of Reciprocal Method Processes

10 A BUDGET

Figures 42a, 42b, 42c and 42d, respectively, are flowcharts of example process control steps performed by a representative example of BUDGET method 2250 provided by the preferred embodiment. In the preferred embodiment, BUDGET method
 15 2250 may operate in any of four different modes:

- use (see Figure 42a)
- administrative request (see Figure 42b)
- administrative response (see Figure 42c)
- administrative reply (see Figure 42d).

20 In general, the "use" mode of BUDGET method 2250 is invoked in response to an event relating to the use of an object or its content. The "administrative request" mode of BUDGET method 2250 is invoked by or on behalf of the user in response to some user action that requires contact with a VDE financial provider,
 25 and basically its task is to send an administrative request to the

VDE financial provider. The "administrative response" mode of BUDGET method 2250 is performed at the VDE financial provider in response to receipt of an administrative request sent from a VDE node to the VDE financial provider by the

5 "administrative request" invocation of BUDGET method 2250 shown in Figure 42b. The "administrative response" invocation of BUDGET method 2250 results in the transmission of an administrative object from VDE financial provider to the VDE user node. Finally, the "administrative reply" invocation of

10 BUDGET method 2250 shown in Figure 42d is performed at the user VDE node upon receipt of the administrative object sent by the "administrative response" invocation of the method shown in Figure 42c.

15 In the preferred embodiment, the same BUDGET method 2250 performs each of the four different step sequences shown in Figures 42a-42d. In the preferred embodiment, different event codes may be passed to the BUDGET method 2250 to invoke these various different modes. Of course, it would be possible to

20 use four separate BUDGET methods instead of a single BUDGET method with four different "dynamic personalities," but the preferred embodiment obtains certain advantages by using the same BUDGET method for each of these four types of invocations.

Looking at Figure 42a, the "use" invocation of BUDGET method 2250 first primes the Budget Audit Trail (blocks 2252, 2254). It then obtains the DTD for the Budget UDE, which it uses to obtain and read the Budget UDE blocks 2256-2262).

5 BUDGET method 2250 in this "use" invocation may then determine whether a Budget Audit date has expired, and terminate if it has ("yes" exit to decision block 2264; blocks 2266, 2268). So long as the Budget Audit date has not expired, the method may then update the Budget using the atomic element
10 and event counts (and possibly other information) (blocks 2270, 2272), and may then save a Budget User Audit record in a Budget Audit Trail UDE (blocks 2274, 2276) before terminating (at terminate point 2278).

15 Looking at Figure 42b, the first six steps (blocks 2280-2290) may be performed by the user VDE node in response to some user action (e.g., request to access new information, request for a new budget, etc.). This "administrative request" invocation of BUDGET method 2250 may prime an audit trail (blocks 2280,
20 2282). The method may then place a request for administrative processing of an appropriate Budget onto a request queue (blocks 2284, 2286). Finally, the method may save appropriate audit trail information (blocks 2288, 2290). Sometime later, the user VDE node may prime a communications audit trail (blocks 2292,

2294), and may then write a Budget Administrative Request into an administrative object (block 2296). This step may obtain information from the secure database as needed from such sources such as, for example, Budget UDE; Budget Audit Trail UDE(s); and Budget Administrative Request Record(s) (block 2298).

Block 2296 may then communicate the administrative object to a VDE financial provider, or alternatively, block 2296 may pass administrative object to a separate communications process or method that arranges for such communications to occur. If desired, method 2250 may then save a communications audit trail (blocks 2300, 2302) before terminating (at termination point 2304).

Figure 42c is a flowchart of an example of process control steps performed by the example of BUDGET method 2250 provided by the preferred embodiment operating in an "administrative response" mode. Steps shown in Figure 42c would, for example, be performed by a VDE financial provider who has received an administrative object containing a Budget administrative request as created (and communicated to a VDE administrator for example) by Figure 42b (block 2296).

Upon receiving the administrative object, BUDGET method 2250 at the VDE financial provider site may prime a budget communications and response audit trail (blocks 2306, 2308), and may then unpack the administrative object and retrieve the budget request(s), audit trail(s) and record(s) it contains (block 2310). This information retrieved from the administrative object may be written by the VDE financial provider into its secure database (block 2312). The VDE financial provider may then retrieve the budget request(s) and determine the response method it needs to execute to process the request (blocks 2314, 2316). BUDGET method 2250 may send the event(s) contained in the request record(s) to the appropriate response method and may generate response records and response requests based on the RESPONSE method (block 2318). The process performed by block 2318 may satisfy the budget request by writing appropriate new response records into the VDE financial provider's secure database (block 2320). BUDGET method 2250 may then write these Budget administrative response records into an administrative object (blocks 2322, 2324), which it may then communicate back to the user node that initiated the budget request. BUDGET method 2250 may then save communications and response processing audit trail information into appropriate audit trail UDE(s) (blocks 2326, 2328) before terminating (at termination point 2330).

Figure 42d is a flowchart of an example of program control steps performed by a representative example of BUDGET method 2250 operating in an "administrative reply" mode. Steps shown in Figure 42d might be performed, for example, by a VDE user node upon receipt of an administrative object containing budget-related information. BUDGET method 2250 may first prime a Budget administrative and communications audit trail (blocks 2332, 2334). BUDGET method 2250 may then extract records and requests from a received administrative object and write the reply record to the VDE secure database (blocks 2336, 2338). The VDE user node may then save budget administrative and communications audit trail information in an appropriate audit trail UDE(s) (blocks 2340, 2341).

Sometime later, the VDE user node may retrieve the reply record from the secure database and determine what method is required to process it (blocks 2344, 2346). The VDE user node may, optionally, prime an audit trail (blocks 2342, 2343) to record the results of the processing of the reply event. The BUDGET method 2250 may then send event(s) contained in the reply record(s) to the REPLY method, and may generate/update the secure database records as necessary to, for example, insert new budget records, delete old budget records and/or apply changes to budget records (blocks 2348, 2350). BUDGET method

2250 may then delete the reply record from the secure data base (blocks 2352, 2353) before writing the audit trail (if required) (blocks 2354m 2355) terminating (at terminate point 2356).

5 **B. REGISTER**

 Figures 43a-43d are flowcharts of an example of program control steps performed by a representative example of a REGISTER method 2400 provided by the preferred embodiment. In this example, the REGISTER method 2400 performs the
10 example steps shown in Figure 43a when operating in a "use" mode, performs the example steps shown in Figure 43b when operating in an "administrative request" mode, performs the steps shown in Figure 43c when operating in an "administrative response" mode, and performs the steps shown in Figure 43d
15 when operating in an "administrative reply" mode.

 The steps shown in Figure 43a may be, for example, performed at a user VDE node in response to some action by or on behalf of the user. For example the user may ask to access an
20 object that has not yet been (or is not now) properly registered to her. In response to such a user request, the REGISTER method 2400 may prime a Register Audit Trail UDE (blocks 2402, 2404) before determining whether the object being requested has already been registered (decision block 2406). If the object has

already been registered ("yes" exit to decision block 2406), the REGISTER method may terminate (at termination point 2408). If the object is not already registered ("no" exit to decision block 2406), then REGISTER method 2400 may access the VDE node
5 secure database PERC 808 and/or Register MDE (block 2410). REGISTER method 2400 may extract an appropriate Register Record Set from this PERC 808 and/or Register MDE (block 2412), and determine whether all of the required elements are present that are needed to register the object (decision block
10 2414). If some piece(s) is missing ("no" exit to decision block 2414), REGISTER method 2400 may queue a Register request record to a communication manager and then suspend the REGISTER method until the queued request is satisfied (blocks 2416, 2418). Block 2416 may have the effect of communicating a
15 register request to a VDE distributor, for example. When the request is satisfied and the register request record has been received (block 2420), then the test of decision block 2414 is satisfied ("yes" exit to decision block 2414), and REGISTER method 2400 may proceed. At this stage, the REGISTER method
20 2400 may allow the user to select Register options from the set of method options allowed by PERC 808 accessed at block 2410 (block 2422). As one simple example, the PERC 808 may permit the user to pay by VISA or MasterCard but not by American Express; block 2422 may display a prompt asking the user to

select between paying using her VISA card and paying using her MasterCard (block 2424). The REGISTER method 2400 preferably validates the user selected registration options and requires the user to select different options if the initial user options were invalid (block 2426, "no" exit to decision block 2428). Once the user has made all required registration option selections and those selections have been validated ("yes" exit to decision block 2428), the REGISTER method 2400 may write an User Registration Table (URT) corresponding to this object and this user which embodies the user registration selections made by the user along with other registration information required by PERC 808 and/or the Register MDE (blocks 2430, 2432). REGISTER method 2400 may then write a Register audit record into the secure database (blocks 2432, 2434) before terminating (at terminate point 2436).

Figure 43b shows an example of an "administrative request" mode of REGISTER method 2400. This Administrative Request Mode may occur on a VDE user system to generate an appropriate administrative object for communication to a VDE distributor or other appropriate VDE participant requesting registration information. Thus, for example, the steps shown in Figure 43b may be performed as part of the "queue register request record" block 2416 shown in Figure 43a. To make a

Register administrative request, REGISTER method 2400 may first prime a communications audit trail (blocks 2440, 2442), and then access the secure database to obtain data about registration (block 2444). This secure database access may, for example,

5 allow the owner and/or publisher of the object being registered to find out demographic, user or other information about the user. As a specific example, suppose that the object being registered is a spreadsheet software program. The distributor of the object may want to know what other software the user has registered.

10 For example, the distributor may be willing to give preferential pricing if the user registers a "suite" of multiple software products distributed by the same distributor. Thus, the sort of information solicited by a "user registration" card enclosed with most standard software packages may be solicited and

15 automatically obtained by the preferred embodiment at registration time. In order to protect the privacy rights of the user, REGISTER method 2400 may pass such user-specific data through a privacy filter that may be at least in part customized by the user so the user can prevent certain information from

20 being revealed to the outside world (block 2446). The REGISTER method 2400 may write the resulting information along with appropriate Register Request information identifying the object and other appropriate parameters into an administrative object (blocks 2448, 2450). REGISTER method

2400 may then pass this administrative object to a communications handler. REGISTER method 2400 may then save a communications audit trail (blocks 2452, 2454) before terminating (at terminate point 2456).

5

Figure 43c includes REGISTER method 2400 steps that may be performed by a VDE distributor node upon receipt of Register Administrative object sent by block 2448, Figure 43b. REGISTER method 2400 in this "administrative response" mode may prime appropriate audit trails (blocks 2460, 2462), and then may unpack the received administrative object and write the associated register request(s) configuration information into the secure database (blocks 2464, 2466). REGISTER method 2400 may then retrieve the administrative request from the secure database and determine which response method to run to process the request (blocks 2468, 2470). If the user fails to provide sufficient information to register the object, REGISTER method 2400 may fail (blocks 2472, 2474). Otherwise, REGISTER method 2400 may send event(s) contained in the appropriate request record(s) to the appropriate response method, and generate and write response records and response requests (e.g., PERC(s) and/or UDEs) to the secure database (blocks 2476, 2478). REGISTER method 2400 may then write the appropriate Register administrative response record into an administrative

10

15

20

object (blocks 2480, 2482). Such information may include, for example, one or more replacement PERC(s) 808, methods, UDE(s), etc. (block 2482). This enables, for example, a distributor to distribute limited right permissions giving users only enough information to register an object, and then later, upon registration, replacing the limited right permissions with wider permissioning scope granting the user more complete access to the objects. REGISTER method 2400 may then save the communications and response processing audit trail (blocks 2484, 2486), before terminating (at terminate point 2488).

Figure 43d shows steps that may be performed by the VDE user node upon receipt of the administrative object generated/transmitted by Figure 43c block 2480. The steps shown in Figure 43d are very similar to those shown in Figure 42d for the BUDGET method administrative reply process.

C. AUDIT

Figures 44a-44c are flowcharts of examples of program control steps performed by a representative example of an AUDIT method 2520 provided by the preferred embodiment. As in the examples above, the AUDIT method 2520 provides three different operational modes in this preferred embodiment example: Figure 44a shows the steps performed by the AUDIT

method in an "administrative request" mode; Figure 44b shows steps performed by the method in the "administrative response" mode; and Figure 44c shows the steps performed by the method in an "administrative reply" mode.

5

The AUDIT method 2520 operating in the "administrative request" mode as shown in Figure 44a is typically performed, for example, at a VDE user node based upon some request by or on behalf of the user. For example, the user may have requested an audit, or a timer may have expired that initiates communication of audit information to a VDE content provider or other VDE participant. In the preferred embodiment, different audits of the same overall process may be performed by different VDE participants. A particular "audit" method 2520 invocation may be initiated for any one (or all) of the involved VDE participants. Upon invocation of AUDIT method 2520, the method may prime an audit administrative audit trail (thus, in the preferred embodiment, the audit processing may itself be audited) (blocks 2522, 2524). The AUDIT method 2520 may then queue a request for administrative processing (blocks 2526, 2528), and then may save the audit administrative audit trail in the secure database (blocks 2530, 2532). Sometime later, AUDIT method 2520 may prime a communications audit trail (blocks 2534, 2536), and may then write Audit Administrative Request(s) into one or more

10

15

20

administrative object(s) based on specific UDE, audit trail UDE(s), and/or administrative record(s) stored in the secure database (blocks 2538, 2540). The AUDIT method 2520 may then save appropriate information into the communications
5 audit trail (blocks 2542, 2544) before terminating (at terminate point 2546).

Figure 44b shows example steps performed by a VDE content provider, financial provider or other auditing VDE node
10 upon receipt of the administrative object generated and communicated by Figure 44a block 2538. The AUDIT method 2520 in this "administrative response" mode may first prime an Audit communications and response audit trail (blocks 2550, 2552), and may then unpack the received administrative object
15 and retrieve its contained Audit request(s) audit trail(s) and audit record(s) for storage into the secured database (blocks 2554, 2556). AUDIT method 2520 may then retrieve the audit request(s) from the secure database and determine the response method to run to process the request (blocks 2558, 2560). AUDIT
20 method 2520 may at this stage send event(s) contained in the request record(s) to the appropriate response method, and generate response record(s) and requests based on this method (blocks 2562, 2564). The processing block 2562 may involve a communication to the outside world.

For example, AUDIT method 2520 at this point could call an external process to perform, for example, an electronic funds transfer against the user's bank account or some other bank account. The AUDIT administrative response can, if desired, call
5 an external process that interfaces VDE to one or more existing computer systems. The external process could be passed the user's account number, PIN, dollar amount, or any other information configured in, or associated with, the VDE audit trail being processed. The external process can communicate
10 with non-VDE hosts and use the information passed to it as part of these communications. For example, the external process could generate automated clearinghouse (ACH) records in a file for submittal to a bank. This mechanism would provide the ability to automatically credit or debit a bank account in any
15 financial institution. The same mechanism could be used to communicate with the existing credit card (e.g. VISA) network by submitting VDE based charges against the charge account.

Once the appropriate Audit response record(s) have been
20 generated, AUDIT method 2520 may write an Audit administrative record(s) into an administrative object for communication back to the VDE user node that generated the Audit request (blocks 2566, 2568). The AUDIT method 2520 may then save communications and response processing audit

information in appropriate audit trail(s) (blocks 2570, 2572)
before terminating (at terminate point 2574).

Figure 44c shows an example of steps that may be
performed by the AUDIT method 2520 back at the VDE user
node upon receipt of the administrative object generated and
sent by Figure 44b, block 2566. The steps 2580-2599 shown in
Figure 44c are similar to the steps shown in Figure 43d for the
REGISTER method 2400 in the "administrative reply" mode.

Briefly, these steps involve receiving and extracting appropriate
response records from the administrative object (block 2584), and
then processing the received information appropriately to update
secure database records and perform any other necessary actions
(blocks 2595, 2596).

Examples of Event-Driven Content-Based Methods

VDE methods 1000 are designed to provide a very flexible
and highly modular approach to secure processing. A complete
VDE process to service a "use event" may typically be
constructed as a combination of methods 1000. As one example,
the typical process for reading content or other information from
an object 300 may involve the following methods:

- an EVENT method
- a METER method

- a BILLING method
- a BUDGET method.

Figure 45 is an example of a sequential series of methods performed by VDE 100 in response to an event. In this example, when an event occurs, an EVENT method 402 may "qualify" the event to determine whether it is significant or not. Not all events are significant. For example, if the EVENT method 1000 in a control process dictates that usage is to be metered based upon number of pages read, then user request "events" for reading less than a page of information may be ignored. In another example, if a system event represents a request to read a certain number of bytes, and the EVENT method 1000 is part of a control process designed to meter paragraphs, then the EVENT method may evaluate the read request to determine how many paragraphs are represented in the bytes requested. This process may involve mapping to "atomic elements" to be discussed in more detail below.

EVENT method 402 filters out events that are not significant with regard to the specific control method involved. EVENT method 402 may pass on qualified events to a METER process 1404, which meters or discards the event based on its own particular criteria.

In addition, the preferred embodiment provides an optimization called "precheck." EVENT method/process 402 may perform this "precheck" based on metering, billing and budget information to determine whether processing based on an event will be allowed. Suppose, for example, that the user has already exceeded her budget with respect to accessing certain information content so that no further access is permitted.

Although BUDGET method 408 could make this determination, records and processes performed by BUDGET method 404 and/or BILLING method 406 might have to be "undone" to, for example, prevent the user from being charged for an access that was actually denied. It may be more efficient to perform a "precheck" within EVENT method 402 so that fewer transactions have to be "undone."

METER method 404 may store an audit record in a meter "trail" UDE 1200, for example, and may also record information related to the event in a meter UDE 1200. For example, METER method 404 may increment or decrement a "meter" value within a meter UDE 1200 each time content is accessed. The two different data structures (meter UDE and meter trail UDE) may be maintained to permit record keeping for reporting purposes to be maintained separately from record keeping for internal operation purposes, for example.

Once the event is metered by METER method 404, the metered event may be processed by a BILLING method 406. BILLING method 406 determines how much budget is consumed by the event, and keeps records that are useful for reconciliation of meters and budgets. Thus, for example, BILLING method 406 may read budget information from a budget UDE, record billing information in a billing UDE, and write one or more audit records in a billing trail UDE. While some billing trail information may duplicate meter and/or budget trail information, the billing trail information is useful, for example, to allow a content creator 102 to expect a payment of a certain size, and serve as a reconciliation check to reconcile meter trail information sent to creator 102 with budget trail information sent to, for example, an independent budget provider.

15

BILLING method 406 may then pass the event on to a BUDGET method 408. BUDGET method 408 sets limits and records transactional information associated with those limits. For example, BUDGET method 408 may store budget information in a budget UDE, and may store an audit record in a budget trail UDE. BUDGET method 408 may result in a "budget remaining" field in a budget UDE being decremented by an amount specified by BILLING method 406.

20

The information content may be released, or other action taken, once the various methods 402, 404, 406, 408 have processed the event.

5 As mentioned above, PERCs 808 in the preferred embodiment may be provided with "control methods" that in effect "oversee" performance of the other required methods in a control process. Figure 46 shows how the required methods/processes 402, 404, 406, and 408 of Figure 45 can be
10 organized and controlled by a control method 410. Control method 410 may call, dispatch events, or otherwise invoke the other methods 402, 404, 406, 408 and otherwise supervise the processing performed in response to an "event."

15 Control methods operate at the level of control sets 906 within PERCs 808. They provide structure, logic, and flow of control between disparate acquired methods 1000. This mechanism permits the content provider to create any desired chain of processing, and also allows the specific chain of
20 processing to be modified (within permitted limits) by downstream redistributors. This control structure concept provides great flexibility.

Figure 47 shows an example of an "aggregate" method 412 which collects METER method 404, BUDGET method 406 and BILLING method 408 into an "aggregate" processing flow. Aggregate method 412 may, for example, combine various elements of metering, budgeting and billing into a single method 1000. Aggregate method 412 may provide increased efficiency as a result of processing METER method 404, BUDGET method 406 and BILLING method 408 aggregately, but may decrease flexibility because of decreased modularity.

10

Many different methods can be in effect simultaneously. Figure 48 shows an example of preferred embodiment event processing using multiple METER methods 404 and multiple BUDGET methods 1408. Some events may be subject to many different required methods operating independently or cumulatively. For example, in the example shown in Figure 48, meter method 404a may maintain meter trail and meter information records that are independent from the meter trail and meter information records maintained by METER method 404b. Similarly, BUDGET method 408a may maintain records independently of those records maintained by BUDGET method 408b. Some events may bypass BILLING method 408 while nevertheless being processed by meter method 404a and

15

20

BUDGET method 408a. A variety of different variations are possible.

REPRESENTATIVE EXAMPLES OF VDE METHODS

5 Although methods 1000 can have virtually unlimited variety and some may even be user-defined, certain basic "use" type methods are preferably used in the preferred embodiment to control most of the more fundamental object manipulation and other functions provided by VDE 100. For example, the
10 following high level methods would typically be provided for object manipulation:

- OPEN method
- READ method
- WRITE method
- 15 • CLOSE method.

 An OPEN method is used to control opening a container so its contents may be accessed. A READ method is used to control the access to contents in a container. A WRITE method is used
20 to control the insertion of contents into a container. A CLOSE method is used to close a container that has been opened.

Subsidiary methods are provided to perform some of the steps required by the OPEN, READ, WRITE and/or CLOSE methods. Such subsidiary methods may include the following:

- ACCESS method
- 5 • PANIC method
- ERROR method
- DECRYPT method
- ENCRYPT method
- DESTROY content method
- 10 • INFORMATION method
- OBSCURE method
- FINGERPRINT method
- EVENT method.
- CONTENT method
- 15 • EXTRACT method
- EMBED method
- METER method
- BUDGET method
- REGISTER method
- 20 • BILLING method
- AUDIT method

An ACCESS method may be used to physically access content associated with an opened container (the content can be

anywhere). A PANIC method may be used to disable at least a portion of the VDE node if a security violation is detected. An ERROR method may be used to handle error conditions. A DECRYPT method is used to decrypt encrypted information. An ENCRYPT method is used to encrypt information. A DESTROY content method is used to destroy the ability to access specific content within a container. An INFORMATION method is used to provide public information about the contents of a container. An OBSCURE method is used to devalue content read from an opened container (e.g., to write the word "SAMPLE" over a displayed image). A FINGERPRINT method is used to mark content to show who has released it from the secure container. An event method is used to convert events into different events for response by other methods.

Open

Figure 49 is a flowchart of an example of preferred embodiment process control steps for an example of an OPEN method 1500. Different OPEN methods provide different detailed steps. However, the OPEN method shown in Figure 49 is a representative example of a relatively full-featured "open" method provided by the preferred embodiment. Figure 49 shows a macroscopic view of the OPEN method. Figures 49a-49f are

together an example of detailed program controlled steps performed to implement the method shown in Figure 49.

The OPEN method process starts with an "open event."

5 This open event may be generated by a user application, an operating system intercept or various other mechanisms for capturing or intercepting control. For example, a user application may issue a request for access to a particular content stored within the VDE container. As another example, another
10 method may issue a command.

In the example shown, the open event is processed by a control method 1502. Control method 1502 may call other methods to process the event. For example, control method 1502
15 may call an EVENT method 1504, a METER method 1506, a BILLING method 1508, and a BUDGET method 1510. Not all OPEN control methods necessarily call of these additional methods, but the OPEN method 1500 shown in Figure 49 is a representative example.

20

Control method 1502 passes a description of the open event to EVENT method 1504. EVENT method 1504 may determine, for example, whether the open event is permitted and whether the open event is significant in the sense that it needs to

be processed by METER method 1506, BILLING method 1508, and/or BUDGET method 1510. EVENT method 1504 may maintain audit trail information within an audit trail UDE, and may determine permissions and significance of the event by
5 using an Event Method Data Element (MDE). EVENT method 1504 may also map the open event into an "atomic element" and count that may be processed by METER method 1506, BILLING method 1508, and/or BUDGET method 1510.

10 In OPEN method 1500, once EVENT method 1504 has been called and returns successfully, control method 1502 then may call METER method 1506 and pass the METER method, the atomic element and count returned by EVENT method 1504. METER method 1506 may maintain audit trail information in a
15 METER method Audit Trail UDE, and may also maintain meter information in a METER method UDE. In the preferred embodiment, METER method 1506 returns a meter value to control method 1502 assuming successful completion.

20 In the preferred embodiment, control method 1502 upon receiving an indication that METER method 1506 has completed successfully, then calls BILLING method 1508. Control method 1502 may pass to BILLING method 1508 the meter value provided by METER method 1506. BILLING method 1508 may

read and update billing information maintained in a BILLING method map MDE, and may also maintain and update audit trail in a BILLING method Audit Trail UDE. BILLING method 1508 may return a billing amount and a completion code to control method 1502.

Assuming BILLING method 1508 completes successfully, control method 1502 may pass the billing value provided by BILLING method 1508 to BUDGET method 1510. BUDGET method 1510 may read and update budget information within a BUDGET method UDE, and may also maintain audit trail information in a BUDGET method Audit Trail UDE. BUDGET method 1510 may return a budget value to control method 1502, and may also return a completion code indicating whether the open event exceeds the user's budget (for this type of event).

Upon completion of BUDGET method 1510, control method 1502 may create a channel and establish read/use control information in preparation for subsequent calls to the READ method.

Figures 49a-49f are a more detailed description of the OPEN method 1500 example shown in Figure 49. Referring to Figure 49a, in response to an open event, control method 1502

first may determine the identification of the object to be opened and the identification of the user that has requested the object to be opened (block 1520). Control method 1502 then determines whether the object to be opened is registered for this user
5 (decision block 1522). It makes this determination at least in part in the preferred embodiment by reading the PERC 808 and the User Rights Table (URT) element associated with the particular object and particular user determined by block 1520 (block 1524). If the user is not registered for this particular
10 object ("no" exit to decision block 1522), then control method 1502 may call the REGISTER method for the object and restart the OPEN method 1500 once registration is complete (block 1526). The REGISTER method block 1526 may be an independent process and may be time independent. It may, for
15 example, take a relatively long time to complete the REGISTER method (say if the VDE distributor or other participant responsible for providing registration wants to perform a credit check on the user before registering the user for this particular object).

20

Assuming the proper URT for this user and object is present such that the object is registered for this user ("yes" exit to decision block 1522), control method 1502 may determine whether the object is already open for this user (decision block

1528). This test may avoid creating a redundant channel for opening an object that is already open. Assuming the object is not already open ("no" exit to decision block 1528), control method 1502 creates a channel and binds appropriate open control elements to it (block 1530). It reads the appropriate open control elements from the secure database (or the container, such as, for example, in the case of a travelling object), and "binds" or "links" these particular appropriate control elements together in order to control opening of the object for this user. Thus, block 1530 associates an event with one or more appropriate method core(s), appropriate load modules, appropriate User Data Elements, and appropriate Method Data Elements read from the secure database (or the container) (block 1532). At this point, control method 1502 specifies the open event (which started the OPEN method to begin with), the object ID and user ID (determined by block 1520), and the channel ID of the channel created by block 1530 to subsequent EVENT method 1504, METER method 1506, BILLING method 1508 and BUDGET method 1510 to provide a secure database "transaction" (block 1536). Before doing so, control method 1502 may prime an audit process (block 1533) and write audit information into an audit UDE (block 1534) so a record of the transaction exists even if the transaction fails or is interfered with.

The detail steps performed by EVENT method 1504 are set forth on Figure 49b. EVENT method 1504 may first prime an event audit trail if required (block 1538) which may write to an EVENT Method Audit Trail UDE (block 1540). EVENT method 5 1504 may then perform the step of mapping the open event to an atomic element number and event count using a map MDE (block 1542). The EVENT method map MDE may be read from the secure database (block 1544). This mapping process performed by block 1542 may, for example, determine whether or 10 not the open event is meterable, billable, or budgetable, and may transform the open event into some discrete atomic element for metering, billing and/or budgeting. As one example, block 1542 might perform a one-to-one mapping between open events and "open" atomic elements, or it may only provide an open atomic 15 element for every fifth time that the object is opened. The map block 1542 preferably returns the open event, the event count, the atomic element number, the object ID, and the user ID. This information may be written to the EVENT method Audit Trail UDE (block 1546, 1548). In the preferred embodiment, a test 20 (decision block 1550) is then performed to determine whether the EVENT method failed. Specifically, decision block 1550 may determine whether an atomic element number was generated. If no atomic element number was generated (e.g., meaning that the open event is not significant for processing by METER method

1506, BILLING method 1508 and/or BUDGET method 1510), then EVENT method 1504 may return a "fail" completion code to control method 1502 ("no" exit to decision block 1550).

5 Control method 1502 tests the completion code returned by
EVENT method 1504 to determine whether it failed or was
successful (decision block 1552). If the EVENT method failed
("no" exit to decision block 1552), control method 1502 may "roll
back" the secure database transaction (block 1554) and return
10 itself with an indication that the OPEN method failed (block
1556). In this context, "rolling back" the secure database
transaction means, for example, "undoing" the changes made to
audit trail UDE by blocks 1540, 1548. However, this "roll back"
performed by block 1554 in the preferred embodiment does not
15 "undo" the changes made to the control method audit UDE by
blocks 1532, 1534.

 Assuming the EVENT method 1504 completed
successfully, control method 1502 then calls the METER method
20 1506 shown on Figure 49c. In the preferred embodiment,
METER method 1506 primes the meter audit trail if required
(block 1558), which typically involves writing to a METER
method audit trail UDE (block 1560). METER method 1506 may
then read a METER method UDE from the secure database

(block 1562), modify the meter UDE by adding an appropriate event count to the meter value contained in the meter UDE (block 1564), and then writing the modified meter UDE back to the secure database (block 1562). In other words, block 1564
5 may read the meter UDE, increment the meter count it contains, and write the changed meter UDE back to the secure database. In the preferred embodiment, METER method 1506 may then write meter audit trail information to the METER method audit trail UDE if required (blocks 1566, 1568). METER method 1506
10 preferably next performs a test to determine whether the meter increment succeeded (decision block 1570). METER method 1506 returns to control method 1502 with a completion code (e.g., succeed or fail) and a meter value determined by block 1564.

15 Control method 1502 tests whether the METER method succeeded by examining the completion code, for example (decision block 1572). If the METER method failed ("no" exit to decision block 1572), then control method 1502 "rolls back" a secure database transaction (block 1574), and returns with an
20 indication that the OPEN method failed (block 1576). Assuming the METER method succeeded ("yes" exit to decision block 1572), control method 1502 calls the BILLING method 1508 and passes it the meter value provided by METER method 1506.

An example of steps performed by BILLING method 1508 is set forth in Figure 49d. BILLING method 1508 may prime a billing audit trail if required (block 1578) by writing to a BILLING method Audit Trail UDE within the secure database (block 1580). BILLING method 1508 may then map the atomic element number, count and meter value to a billing amount using a BILLING method map MDE read from the secure database (blocks 1582, 1584). Providing an independent BILLING method map MDE containing, for example, price list information, allows separately deliverable pricing for the billing process. The resulting billing amount generated by block 1582 may be written to the BILLING method Audit Trail UDE (blocks 1586, 1588), and may also be returned to control method 1502. In addition, BILLING method 1508 may determine whether a billing amount was properly selected by block 1582 (decision block 1590). In this example, the test performed by block 1590 generally requires more than mere examination of the returned billing amount, since the billing amount may be changed in unpredictable ways as specified by BILLING method map MDE. Control then returns to control method 1502, which tests the completion code provided by BILLING method 1508 to determine whether the BILLING method succeeded or failed (block 1592). If the BILLING method failed ("no" exit to decision block 1592), control method 1502 may "roll back" the secure database

transaction (block 1594), and return an indication that the
OPEN method failed (block 1596). Assuming the test performed
by decision block 1592 indicates that the BILLING method
succeeded ("yes" exit to decision block 1592), then control method
5 1502 may call BUDGET method 1510.

Other BILLING methods may use site, user and/or usage
information to establish, for example, pricing information. For
example, information concerning the presence or absence of an
10 object may be used in establishing "suite" purchases, competitive
discounts, etc. Usage levels may be factored into a BILLING
method to establish price breaks for different levels of usage. A
currency translation feature of a BILLING method may allow
purchases and/or pricing in many different currencies. Many
15 other possibilities exist for determining an amount of budget
consumed by an event that may be incorporated into BILLING
methods.

An example of detailed control steps performed by
20 BUDGET method 1510 is set forth in Figure 49e. BUDGET
method 1510 may prime a budget audit trail if required by
writing to a budget trail UDE (blocks 1598, 1600). BUDGET
method 1510 may next perform a billing operation by adding a
billing amount to a budget value (block 1602). This operation

may be performed, for example, by reading a BUDGET method UDE from the secure database, modifying it, and writing it back to the secure database (block 1604). BUDGET method 1510 may then write the budget audit trail information to the BUDGET method Audit Trail UDE (blocks 1606, 1608). BUDGET method 1510 may finally, in this example, determine whether the user has run out of budget by determining whether the budget value calculated by block 1602 is out of range (decision block 1610). If the user has run out of budget ("yes" exit to decision block 1610), the BUDGET method 1510 may return a "fail completion" code to control method 1502. BUDGET method 1510 then returns to control method 1502, which tests whether the BUDGET method completion code was successful (decision block 1612). If the BUDGET method failed ("no" exit to decision block 1612), control method 1502 may "roll back" the secure database transaction and itself return with an indication that the OPEN method failed (blocks 1614, 1616). Assuming control method 1502 determines that the BUDGET method was successful, the control method may perform the additional steps shown on Figure 49f. For example, control method 1502 may write an open audit trail if required by writing audit information to the audit UDE that was primed at block 1532 (blocks 1618, 1620). Control method 1502 may then establish a read event processing (block 1622), using the User Right Table and the PERC associated with the object

and user to establish the channel (block 1624). This channel may optionally be shared between users of the VDE node 600, or may be used only by a specified user.

5 Control method 1502 then, in the preferred embodiment, tests whether the read channel was established successfully (decision block 1626). If the read channel was not successfully established ("no" exit to decision block 1626), control method 1502 "rolls back" the secured database transaction and provides
10 an indication that the OPEN method failed (blocks 1628, 1630). Assuming the read channel was successfully established ("yes" exit to decision block 1626), control method 1502 may "commit" the secure database transaction (block 1632). This step of "committing" the secure database transaction in the preferred
15 embodiment involves, for example, deleting intermediate values associated with the secure transaction that has just been performed and, in one example, writing changed UDEs and MDEs to the secure database. It is generally not possible to "roll back" a secure transaction once it has been committed by block
20 1632. Then, control method 1502 may "tear down" the channel for open processing (block 1634) before terminating (block 1636). In some arrangements, such as multi-tasking VDE node environments, the open channel may be constantly maintained and available for use by any OPEN method that starts. In other

implementations, the channel for open processing may be rebuilt and restarted each time an OPEN method starts.

Read

5 Figure 50, 50a-50f show examples of process control steps for performing a representative example of a READ method 1650. Comparing Figure 50 with Figure 49 reveals that the same overall high level processing may typically be performed for
10 READ method 1650 as was described in connection with OPEN method 1500. Thus, READ method 1650 may call a control method 1652 in response to a read event, the control method in turn invoking an EVENT method 1654, a METER method 1656, a BILLING method 1658 and a BUDGET method 1660. In the preferred embodiment, READ control method 1652 may request
15 methods to fingerprint and/or obscure content before releasing the decrypted content.

 Figures 50a-50e are similar to Figures 49a-49e. Of course, even though the same user data elements may be used for both
20 the OPEN method 1500 and the READ method 1650, the method data elements for the READ method may be completely different, and in addition, the user data elements may provide different auditing, metering, billing and/or budgeting criteria for read as opposed to open processing.

Referring to Figure 50f, the READ control method 1652 must determine which key to use to decrypt content if it is going to release decrypted content to the user (block 1758). READ control method 1652 may make this key determination based, in part, upon the PERC 808 for the object (block 1760). READ control method 1652 may then call an ACCESS method to actually obtain the encrypted content to be decrypted (block 1762). The content is then decrypted using the key determined by block 1758 (block 1764). READ control method 1652 may then determine whether a "fingerprint" is desired (decision block 1766). If fingerprinting of the content is desired ("yes" exit of decision block 1766), READ control method 1652 may call the FINGERPRINT method (block 1768). Otherwise, READ control method 1652 may determine whether it is desired to obscure the decrypted content (decision block 1770). If so, READ control method 1652 may call an OBSCURE method to perform this function (block 1772). Finally, READ control method 1652 may commit the secure database transaction (block 1774), optionally tear down the read channel (not shown), and terminate (block 1776).

Write

Figures 51, 51a-51f are flowcharts of examples of process control steps used to perform a representative example of a

WRITE method 1780 in the preferred embodiment. WRITE method 1780 uses a control method 1782 to call an EVENT method 1784, METER method 1786, BILLING method 1788, and BUDGET method 1790 in this example. Thus, writing
5 information into a container (either by overwriting information already stored in the container or adding new information to the container) in the preferred embodiment may be metered, billed and/or budgeted in a manner similar to the way opening a container and reading from a container can be metered, billed
10 and budgeted. As shown in Figure 51, the end result of WRITE method 1780 is typically to encrypt content, update the container table of contents and related information to reflect the new content, and write the content to the object.

15 Figure 51a for the WRITE control method 1782 is similar to Figure 49a and Figure 50a for the OPEN control method and the READ control method, respectively. However, Figure 51b is slightly different from its open and read counterparts. In particular, block 1820 is performed if the WRITE EVENT
20 method 1784 fails. This block 1820 updates the EVENT method map MDE to reflect new data. This is necessary to allow information written by block 1810 to be read by Figure 51b READ method block 1678 based on the same (but now updated) EVENT method map MDE.

Looking at Figure 51f, once the EVENT, METER, BILLING and BUDGET methods have returned successfully to WRITE control method 1782, the WRITE control method writes audit information to Audit UDE (blocks 1890, 1892), and then
5 determines (based on the PERC for the object and user and an optional algorithm) which key should be used to encrypt the content before it is written to the container (blocks 1894, 1896). CONTROL method 1782 then encrypts the content (block 1898) possibly by calling an ENCRYPT method, and writes the
10 encrypted content to the object (block 1900). CONTROL method 1782 may then update the table of contents (and related information) for the container to reflect the newly written information (block 1902), commit the secure database transaction (block 1904), and return (block 1906).

15

Close

Figure 52 is a flowchart of an example of process control steps to perform a representative example of a CLOSE method 1920 in the preferred embodiment. CLOSE method 1920 is used
20 to close an open object. In the preferred embodiment, CLOSE method 1920 primes an audit trail and writes audit information to an Audit UDE (blocks 1922, 1924). CLOSE method 1920 then may destroy the current channel(s) being used to support and/or process one or more open objects (block 1926). As discussed

above, in some (e.g., multi-user or multi-tasking) installations, the step of destroying a channel is not needed because the channel may be left operating for processing additional objects for the same or different users. CLOSE method 1920 also
5 releases appropriate records and resources associated with the object at this time (block 1926). The CLOSE method 1920 may then write an audit trail (if required) into an Audit UDE (blocks 1928, 1930) before completing.

10 **Event**

Figure 53a is a flowchart of example process control steps provided by a more general example of an EVENT method 1940 provided by the preferred embodiment. Examples of EVENT methods are set forth in Figures 49b, 50b and 51b and are
15 described above. EVENT method 1940 shown in Figure 53a is somewhat more generalized than the examples above. Like the EVENT method examples above, EVENT method 1940 receives an identification of the event along with an event count and event parameters. EVENT method 1940 may first prime an
20 EVENT audit trail (if required) by writing appropriate information to an EVENT method Audit Trail UDE (blocks 1942, 1944). EVENT method 1940 may then obtain and load an EVENT method map DTD from the secure database (blocks 1946, 1948). This EVENT method map DTD describes, in this

example, the format of the EVENT method map MDE to be read and accessed immediately subsequently (by blocks 1950, 1952).

In the preferred embodiment, MDEs and UDEs may have any of various different formats, and their formats may be flexibly

5 specified or changed dynamically depending upon the installation, user, etc. The DTD, in effect, describes to the EVENT method 1940 how to read from the EVENT method map MDE. DTDs are also used to specify how methods should write to MDEs and UDEs, and thus may be used to implement privacy
10 filters by, for example, preventing certain confidential user information from being written to data structures that will be reported to third parties.

Block 1950 ("map event to atomic element # and event
15 count using a Map MDE") is in some sense the "heart" of EVENT method 1940. This step "maps" the event into an "atomic element number" to be responded to by subsequently called methods. An example of process control steps performed by a somewhat representative example of this "mapping" step 1950 is
20 shown in Figure 53b.

The Figure 53b example shows the process of converting a READ event that is associated with requesting byte range 1001-1500 from a specific piece of content into an appropriate atomic

element. The example EVENT method mapping process (block 1950 in Figure 53a) can be detailed as the representative process shown in Figure 53b.

5 EVENT method mapping process 1950 may first look up
the event code (READ) in the EVENT method MDE (1952) using
the EVENT method map DTD (1948) to determine the structure
and contents of the MDE. A test might then be performed to
determine if the event code was found in the MDE (1956), and if
10 not ("No" branch), the EVENT method mapping process may the
terminate (1958) without mapping the event to an atomic
element number and count. If the event was found in the MDE
("Yes" branch), the EVENT method mapping process may then
compare the event range (e.g., bytes 1001-1500) against the
15 atomic element to event range mapping table stored in the MDE
(block 1960). The comparison might yield one or more atomic
element numbers or the event range might not be found in the
mapping table. The result of the comparison might then be
tested (block 1962) to determine if any atomic element numbers
20 were found in the table. If not ("No" branch), the EVENT
method mapping process may terminate without selecting any
atomic element numbers or counts (1964). If the atomic element
numbers were found, the process might then calculate the atomic
element count from the event range (1966). In this example, the

process might calculate the number of bytes requested by subtracting the upper byte range from the lower byte range (e.g., $1500 - 1001 + 1 = 500$). The example EVENT method mapping process might then terminate (block 1968) and return the atomic element number(s) and counts.

EVENT method 1940 may then write an EVENT audit trail if required to an EVENT method Audit Trail UDE (block 1970, 1972). EVENT method 1940 may then prepare to pass the atomic element number and event count to the calling CONTROL method (or other control process) (at exit point 1978). Before that, however, EVENT method 1940 may test whether an atomic element was selected (decision block 1974). If no atomic element was selected, then the EVENT method may be failed (block 1974). This may occur for a number of reasons. For example, the EVENT method may fail to map an event into an atomic element if the user is not authorized to access the specific areas of content that the EVENT method MDE does not describe. This mechanism could be used, for example, to distribute customized versions of a piece of content and control access to the various versions in the content object by altering the EVENT method MDE delivered to the user. A specific use of this technology might be to control the distribution of different

language (e.g., English, French, Spanish) versions of a piece of content.

Billing

5 Figure 53c is a flowchart of an example of process control steps performed by a BILLING method 1980. Examples of BILLING methods are set forth in Figures 49d, 50d, and 51d and are described above. BILLING method 1980 shown in Figure 53c is somewhat more generalized than the examples above. Like
10 the BILLING method examples above, BILLING method 1980 receives a meter value to determine the amount to bill. BILLING method 1980 may first prime a BILLING audit trail (if required) by writing appropriate information to the BILLING method Audit Trail UDE (blocks 1982, 1984). BILLING method
15 1980 may then obtain and load a BILLING method map DTD from the secure database (blocks 1985, 1986), which describes the BILLING method map MDE (e.g., a price list, table, or parameters to the billing amount calculation algorithm) that should be used by this BILLING method. The BILLING method
20 map MDE may be delivered either as part of the content object or as a separately deliverable component that is combined with the control information at registration.

The BILLING method map MDE in this example may describe the pricing algorithm that should be used in this BILLING method (e.g., bill \$0.001 per byte of content released). Block 1988 ("Map meter value to billing amount") functions in the same manner as block 1950 of the EVENT method; it maps the meter value to a billing value. Process step 1988 may also interrogate the secure database (as limited by the privacy filter) to determine if other objects or information (e.g., user information) are present as part of the BILLING method algorithm.

BILLING method 1980 may then write a BILLING audit trail if required to a BILLING method Audit Trail UDE (block 1990, 1992), and may prepare to return the billing amount to the calling CONTROL method (or other control process). Before that, however, BILLING method 1980 may test whether a billing amount was determined (decision block 1994). If no billing amount was determined, then the BILLING method may be failed (block 1996). This may occur if the user is not authorized to access the specific areas of the pricing table that the BILLING method MDE describes (e.g., you may purchase not more than \$100.00 of information from this content object).

Access

Figure 54 is a flowchart of an example of program control steps performed by an ACCESS method 2000. As described above, an ACCESS method may be used to access content
5 embedded in an object 300 so it can be written to, read from, or otherwise manipulated or processed. In many cases, the ACCESS method may be relatively trivial since the object may, for example, be stored in a local storage that is easily accessible. However, in the general case, an ACCESS method 2000 must go
10 through a more complicated procedure in order to obtain the object. For example, some objects (or parts of objects) may only be available at remote sites or may be provided in the form of a real-time download or feed (e.g., in the case of broadcast transmissions). Even if the object is stored locally to the VDE
15 node, it may be stored as a secure or protected object so that it is not directly accessible to a calling process. ACCESS method 2000 establishes the connections, routings, and security requisites needed to access the object. These steps may be performed transparently to the calling process so that the calling
20 process only needs to issue an access request and the particular ACCESS method corresponding to the object or class of objects handles all of the details and logistics involved in actually accessing the object.

ACCESS method 2000 may first prime an ACCESS audit trail (if required) by writing to an ACCESS Audit Trail UDE (blocks 2002, 2004). ACCESS method 2000 may then read and load an ACCESS method DTD in order to determine the format of an ACCESS MDE (blocks 2006, 2008). The ACCESS method MDE specifies the source and routing information for the particular object to be accessed in the preferred embodiment. Using the ACCESS method DTD, ACCESS method 2000 may load the correction parameters (e.g., by telephone number, account ID, password and/or a request script in the remote resource dependent language).

ACCESS method 2000 reads the ACCESS method MDE from the secure database, reads it in accordance with the ACCESS method DTD, and loads encrypted content source and routing information based on the MDE (blocks 2010, 2012). This source and routing information specifies the location of the encrypted content. ACCESS method 2000 then determines whether a connection to the content is available (decision block 2014). This "connection" could be, for example, an on-line connection to a remote site, a real-time information feed, or a path to a secure/protected resource, for example. If the connection to the content is not currently available ("No" exit of decision block 2014), then ACCESS method 2000 takes steps to

open the connection (block 2016). If the connection fails (e.g., because the user is not authorized to access a protected secure resource), then the ACCESS method 2000 returns with a failure indication (termination point 2018). If the open connection
5 succeeds, on the other hand, then ACCESS method 2000 obtains the encrypted content (block 2020). ACCESS method 2000 then writes an ACCESS audit trail if required to the secure database ACCESS method Audit Trail UDE (blocks 2022, 2024), and then terminates (terminate point 2026).

10

Decrypt and Encrypt

Figure 55a is a flowchart of an example of process control steps performed by a representative example of a DECRYPT method 2030 provided by the preferred embodiment. DECRYPT
15 method 2030 in the preferred embodiment obtains or derives a decryption key from an appropriate PERC 808, and uses it to decrypt a block of encrypted content. DECRYPT method 2030 is passed a block of encrypted content or a pointer to where the encrypted block is stored. DECRYPT 2030 selects a key number
20 from a key block (block 2032). For security purposes, a content object may be encrypted with more than one key. For example, a movie may have the first 10 minutes encrypted using a first key, the second 10 minutes encrypted with a second key, and so on. These keys are stored in a PERC 808 in a structure called a "key

block." The selection process involves determining the correct key to use from the key block in order to decrypt the content. The process for this selection is similar to the process used by EVENT methods to map events into atomic element numbers.

5 DECRYPT method 2030 may then access an appropriate PERC 808 from the secure database 610 and loads a key (or "seed") from a PERC (blocks 2034, 2036). This key information may be the actual decryption key to be used to decrypt the content, or it may be information from which the decryption key may be at

10 least in part derived or calculated. If necessary, DECRYPT method 2030 computes the decryption key based on the information read from PERC 808 at block 2034 (block 2038). DECRYPT method 2030 then uses the obtained and/or calculated decryption key to actually decrypt the block of encrypted

15 information (block 2040). DECRYPT method 2030 outputs the decrypted block (or the pointer indicating where it may be found), and terminates (termination point 2042).

Figure 55b is a flowchart of an example of process control

20 steps performed by a representative example of an ENCRYPT method 2050. ENCRYPT method 2050 is passed as an input, a block of information to encrypt (or a pointer indicating where it may be found). ENCRYPT method 2050 then may determine an encryption key to use from a key block (block 2052). The

encryption key selection makes a determination if a key for a specific block of content to be written already exists in a key block stored in PERC 808. If the key already exists in the key block, then the appropriate key number is selected. If no such

5 key exists in the key block, a new key is calculated using an algorithm appropriate to the encryption algorithm. This key is then stored in the key block of PERC 808 so that DECRYPT method 2030 may access the key in order to decrypt the content stored in the content object. ENCRYPT method 2050 then

10 accesses the appropriate PERC to obtain, derive and/or compute an encryption key to be used to encrypt the information block (blocks 2054, 2056, 2058, which are similar to Figure 55a blocks 2034, 2036, 2038). ENCRYPT method 2050 then actually encrypts the information block using the obtained and/or derived

15 encryption key (block 2060) and outputs the encrypted information block or a pointer where it can be found before terminating (termination point 2062).

Content

20 Figure 56 is a flowchart of an example of process control steps performed by a representative of a CONTENT method 2070 provided by the preferred embodiment. CONTENT method 2070 in the preferred embodiment builds a "synopsis" of protected content using a secure process. For example,

CONTENT method 2070 may be used to derive unsecure
("public") information from secure content. Such derived public
information might include, for example, an abstract, an index, a
table of contents, a directory of files, a schedule when content
5 may be available, or excerpts such as for example, a movie
"trailer."

CONTENT method 2070 begins by determining whether
the derived content to be provided must be derived from secure
10 contents, or whether it is already available in the object in the
form of static values (decision block 2070). Some objects may, for
example, contain prestored abstracts, indexes, tables of contents,
etc., provided expressly for the purpose of being extracted by the
CONTENT method 2070. If the object contains such static
15 values ("static" exit to decision block 2072), then CONTENT
method 2070 may simply read this static value content
information from the object (block 2074), optionally decrypt, and
release this content description (block 2076). If, on the other
hand, CONTENT method 2070 must derive the synopsis/content
20 description from the secure object ("derived" exit to decision block
2072), then the CONTENT method may then securely read
information from the container according to a synopsis algorithm
to produce the synopsis (block 2078).

Extract and Embed

Figure 57a is a flowchart of an example of process control steps performed by a representative example of an EXTRACT method 2080 provided by the preferred embodiment. EXTRACT method 2080 is used to copy or remove content from an object and place it into a new object. In the preferred embodiment, the EXTRACT method 2080 does not involve any release of content, but rather simply takes content from one container and places it into another container, both of which may be secure. Extraction of content differs from release in that the content is never exposed outside a secure container. Extraction and Embedding are complementary functions; extract takes content from a container and creates a new container containing the extracted content and any specified control information associated with that content. Embedding takes content that is already in a container and stores it (or the complete object) in another container directly and/or by reference, integrating the control information associated with existing content with those of the new content.

EXTRACT method 2080 begins by priming an Audit UDE (blocks 2082, 2084). EXTRACT method then calls a BUDGET method to make sure that the user has enough budget for (and is authorized to) extract content from the original object (block

2086). If the user's budget does not permit the extraction ("no" exit to decision block 2088), then EXTRACT method 2080 may write a failure audit record (block 2090), and terminate (termination point 2092). If the user's budget permits the extraction ("yes" exit to decision block 2088), then the EXTRACT method 2080 creates a copy of the extracted object with specified rules and control information (block 2094). In the preferred embodiment, this step involves calling a method that actually controls the copy. This step may or may not involve decryption and encryption, depending on the particular the PERC 808 associated with the original object, for example. EXTRACT method 2080 then checks whether any control changes are permitted by the rights authorizing the extract to begin with (decision block 2096). In some cases, the extract rights require an exact copy of the PERC 808 associated with the original object (or a PERC included for this purpose) to be placed in the new (destination) container ("no" exit to decision block 2096). If no control changes are permitted, then extract method 2080 may simply write audit information to the Audit UDE (blocks 2098, 2100) before terminating (terminate point 2102). If, on the other hand, the extract rights permit the user to make control changes ("yes" to decision block 2096), then EXTRACT method 2080 may call a method or load module that solicits new or changed control information (e.g., from the user, the distributor who